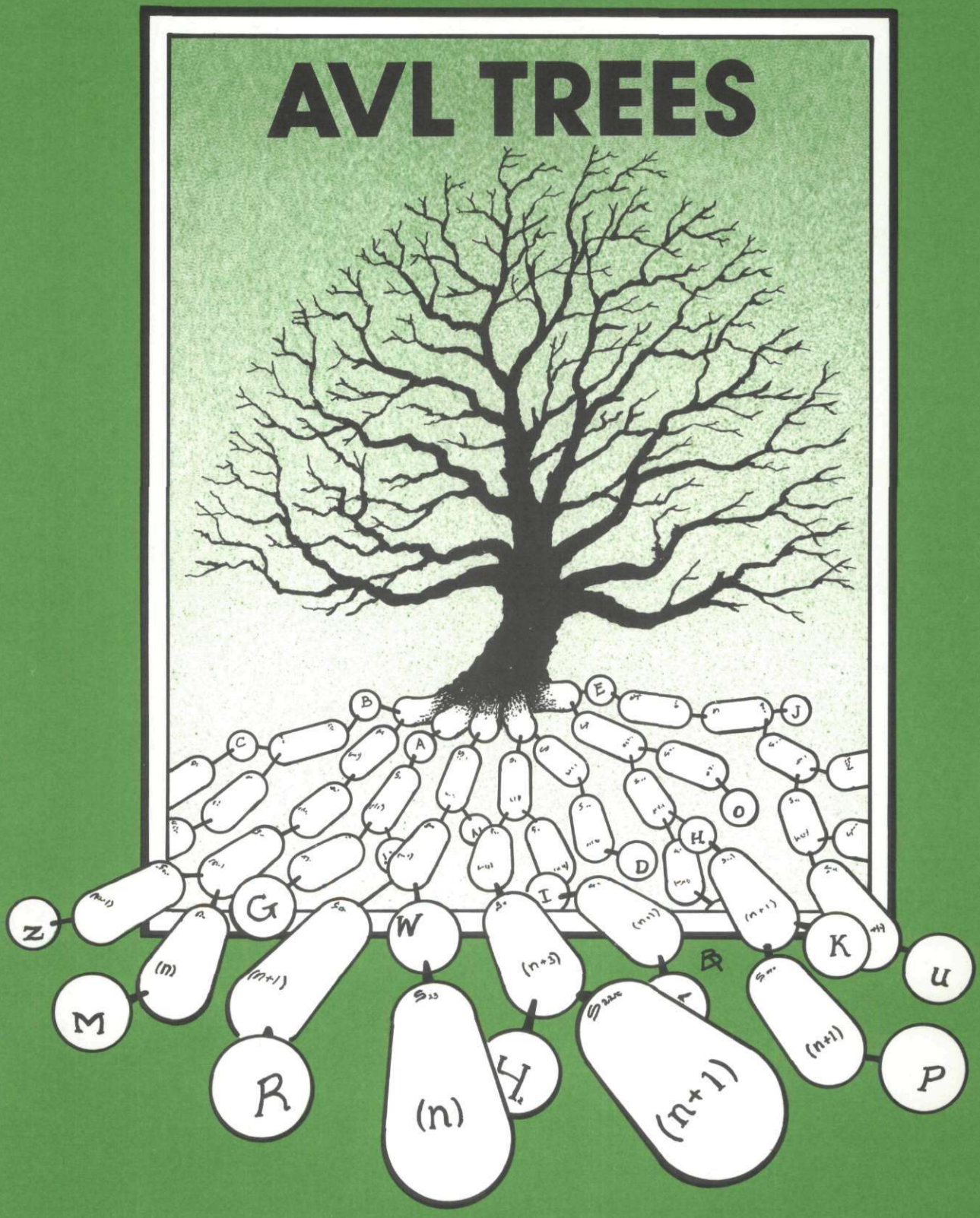
Vol. 4, No. 1 January/February 1986 $4.00

programmer's journal

The Resource Journal for IBM PC Programmers



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Good news for software developers:

No royalties on Btrieve®.

Effective January 1, SoftCraft will no longer charge royalties for our Btrieve file management system.

By dropping royalties, we're giving software authors and application developers something to bank on: increased profitability on every copy of a Btrieve application that is used or sold. What you do with Btrieve is up to you. Whether you're developing applications for a handful of users or hundreds, you pay for Btrieve only once—no strings attached. The price remains the same.

With no royalties, doing business with SoftCraft is easier and more profitable than ever. We're showing our appreciation to current Btrieve users, while furthering the rapid expansion of Btrieve applications. We believe this move will reinforce Btrieve's position as the file management standard for IBM PC or AT software developers.

SoftCraft is committed to providing our customers with the comprehensive development tools they need. In the last 18 months alone we've introduced database query and report writing modules for Btrieve applications, as well as local area network and XENIX versions of Btrieve. Our Btrieve environment continues to grow, keeping you in the forefront as future trends emerge.

We're convinced our new "no more royalties" plan will make you—and Btrieve—even more successful.

And that's a great way to start the new business year.

SoftCraft Inc.

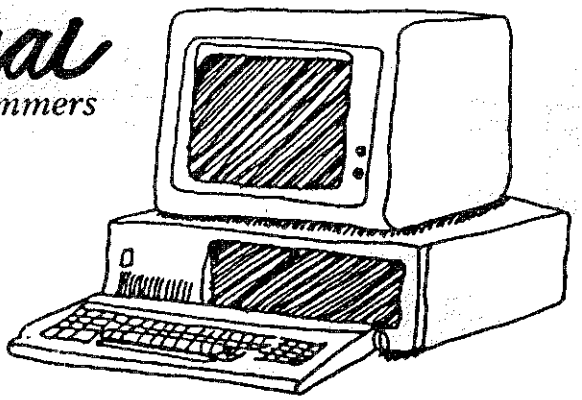
P.O. Box 9802 #917
Austin, Texas 78766
(512) 346-8380 Telex 358 200

Suggested retail prices: Btrieve, $245; Btrieve/N (network version), $595; Xtrieve, $195; Xtrieve/N, $395; Rtrieve, $85; Rtrieve/N, $175.
Requires PC-DOS, MS-DOS 1.X-3.X or XENIX.
Btrieve is a registered trademark and Xtrieve and Rtrieve are trademarks of SoftCraft Inc.

Circle **21** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers



Volume 4, Number 1 January/February 1986

CONTENTS

NOTE FROM THE EDITOR 2

LETTERS 3

PJ FORUM: PC/AT HARD DISK PROBLEMS AND PLUNGING 64K DRAM PRICES by FNE 5
Excerpts from DTACK Grounded: The 68000/32081 Systems Journal
And a response to Peter Norton's letter about compilers in the last issue of PJ

C: AUTOINC EXPLAINED by Randall B. Swan 10
Evaluating the auto-increment with respect to addressing and dereferencing

C: SORTING USING AN AVL TREE by Robert Ackerman 14

PASCAL: COMBINING BINARY TREES by John Mowat 23

PASCAL: USING DOS TO CALL PROGRAMS FROM TURBO PASCAL by Arvind Kumar 31
Also: file control blocks, COMSPEC, error messages, and invoking command.com

AN OPERATING SYSTEM FOR THE PEOPLE by Hal W. Hardenbergh 39
The editor of DTACK Grounded *debuts in* PJ *under his own name*

dBASE II: THE FINAL INSTALLMENT OF BRETT SALTER'S dTOOLS 42

REVIEW: TWO MICROCOMPUTER GRAPHICS BOOKS by J. Edward Volkstorf, Jr. 46

MISC. 49

PJ BILLBOARD 52

HOUSEKEEPING 53

READER SURVEY 56

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NOTE FROM THE EDITOR

programmer's journal

Publisher:	Elizabeth Oakley
Editor:	Robert Keller
Managing Editor:	Erin O'Connor
Production Manager:	Ralph Bentley
Circulation/Promotion:	Robert Tappan
Cover:	Ralph Bentley
Illustrations:	Julia King
Production Assistance:	Holly Tracy

Contributing Writers:

Robert Ackerman	John Mowat
Arvind Kumar	Brett Salter
J. Edward Volkstorf, Jr.	Randall B. Swan
Hal W. Hardenbergh	FNE

Founder: Greg Estes

Business and Editorial Offices:	Programmer's Journal 29 West 29th Ave., Suite 5 Eugene, OR 97405 or P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07843 (503) 484-2162

Subscriptions: Bi-monthly rates $24 U.S.
($30 in Canada and $40 elsewhere)
2 years $44 U.S.
($55 in Canada and $75 elsewhere)
3 years $64 U.S.
($80 in Canada and $110 elsewhere)

Articles in Programmer's Journal represent the views of their authors and are not necessarily those of the publisher.

Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley or Robin Tappan, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403

IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.

Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Application to mail at second-class rates is pending at Eugene, OR. POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834.

© 1986

At the back of this issue of *PJ* (and in last issue as well) is a READER SURVEY, complete with return postage paid. We'll publish the results in these pages. A preliminary result: The age of respondees so far is within 0-75.

Not much housekeeping this issue. Neither is the laundry done, nor bills paid. A software project is almost done, 140K and 1 1/2 years later. The marketing company has stopped calling every week, maybe sensing they were becoming a little obsessed. My daydreams are of trimming XT fan blades.

Bob Ackerman has a C article in this issue on sorting English words with an AVL tree, and John Mowat follows with a tree implementation in Pascal.

Check the *errata* at the back of the magazine for some missing code from last issue's article by Neil Rubenking on Turbo Pascal 3.0 and subdirectories.

This letter from a reader:

I've just sent off payment to Denville for a 3-year renewal.

In these uncertain times, I hope you'll see that as a vote of confidence in your work.

Now that "FNE" is no longer publishing the DTACK Grounded *newsletter (which I learned about through* PJ *and enjoyed very much), do you think you could draft him to write an occasional column? Perhaps a news-of-the-industry type thing?*

Keep up the good work.
Neal Fellers
Creeley, CO 80631

We take this as a vote for FNE as well. He debuts under his own name this month with the article "An Operating System for the People."

—Robert □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

Letters to the editor are considered to be unconditionally assigned for publication and copyright purposes. Programmer's Journal *may edit letters prior to their publication. Opinions expressed do not necessarily represent those of the publisher or editors of* Programmer's Journal.

VARIANT RECORDS

I was somewhat disturbed by the "Turbo Pascal 3.0 and Subdirectories" article in your November/December 1985, issue. What bothered me was that though Neil Rubenking starts out with the statement "TURBO Pascal 3.0 brings TURBO into the world of subdirectories. . ." he then goes on to explain how he used "shl 8" to shift a byte of data into the "AH" byte of "AX" (of the record variable "registers"). Now, had he read the Turbo 3.0 manual carefully, he would have noted that the type "regpack" appears there as a variant record type (as compared to the nonvariant type it was in the Turbo 1.0 and 2.0 manuals) given as

```
TYPE regpack = RECORD CASE
                   INTEGER OF
                   1:(ax,bx,cx,dx,bp,
                      si,di,ds,es,flags:
                      INTEGER);
                   2:(al,ah,bl,bh,cl,
                      ch,dl,dh:byte);
               END;
```

(*Note that when broken up into bytes, the registers are listed as al, ah because of Intel's byte reversed storage format*)

If we then declare the variable register to be of type regpack, we can set AH to be (as an example) $1A by

```
register.ah := $1A;
```

as opposed to the way Neil chose to do it:

```
register.ax := $1A shl 8;
```

Actually, variant records do take a bit of getting used to — I must admit that I just recently learned to use variant records to get rid of some of what I had thought was required assembly language (inserted inline in Pascal source code for a PDP-11). The problem was, I had to use the output of the compiler to figure out what addresses to use in the inline assembly code, which meant that the code was basically not portable — the same assembly code might not necessarily work correctly with different versions of the Pascal compiler. This is not a good way of doing things. After learning about variant records, I figured out a better way to do things. First, let me explain what I was trying to do:

I wished to have a file of real numbers but, as a header to the file, I wished to store characters and integers. As a way to have access to the real number (for the PDP-11, a 4-byte real) as integers and chars, I used the following record type:

```
TYPE rintc = RECORD CASE INTEGER
OF
  1:(r:REAL);          {Takes up 4 bytes}
  2:(i:ARRAY[1..2] of INTEGER);
                       {Takes up 4 bytes}
  3:(c:ARRAY[1..4] of CHAR);
                       {Takes up 4 bytes}
END;

VAR r:rintc;
```

Then you can see that to stuff 2 integers into a real number, all I need to do is:

```
r.i[1]:=int1;
r.i[2]:=int2;
```

Then I can access the real number as r.r — a very straightforward conversion as long as you know the storage size of a real number.

Now, the only thing left somewhat confusing is that regpack uses 2 different sized pieces for its variant part. The way that variant records are set up, the largest portion of the variant determines the size allotted for the variant — the smaller portions of the variant only partially "fill up" the space allotted for the variant. This explains how we can "overlay" al, ah, etc atop ax, bx, cx, and dx.

Sorry about going on for so long about this, but I felt that the way presented in the listings in the article was not the best possible way to do things. I felt obliged to offer my thoughts on the subject along with a brief explanation of how I had come to appreciate the use of the variant record.

Roderick K. Graves
Sunnyvale, CA 94086

APOLOGIES

My letter concerning FORTRAN compilers that was printed in your November issue contained two paragraphs that I very deeply regret having written.

I apologize to MICROSOFT and their many satisfied customers. I recently had the opportunity and great pleasure to work closely with the WINDOWS development group. Their competence, conscientiousness, and dedication to quality is outstanding.

Douglas E. Lippincott
Stanton, CA 90680

Famous Computer Personalities #2: Ramses II, the first dynamic RAM

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Top Quality Programming Tools from the Developers of the TurboPower Utilities

NEW!

Turbo EXTENDER™

Tired of fighting 64K Code and Data Segments?
Bored while waiting for your 10,000 liner to Compile?
Want to optimize those sluggish Overlays?

LARGE CODE MODEL

Write Turbo Pascal programs using all 640K of MSDOS memory, based on any number of separately compiled modules. Provides complete parameter passing using normal Pascal syntax. Heap and Data Segment are shared between all modules. No memory-resident kludges or unnatural parameter passing schemes. Comes with a utility which automatically converts your existing applications.

LARGE DATA ARRAYS

Transparently access 1 and 2 dimensional arrays of any conceivable size and type. Four models support Normal RAM to 640K, Expanded memory (EMS) to 2Meg, Virtual (Disk-based) to 30Meg, and sparse arrays like the most advanced spreadsheets. Comes with a fast full-screen array browser.

MAKE FACILITY

A Unix-like MAKE program that is optimized for the Turbo EXTENDER large code model. Rebuild multi-module programs with no wasted effort.

OVERLAY ANALYST

Perform Static and Dynamic analysis of overlayed Turbo programs. Determine sizes of all procedures in each overlay group. Monitor the running program to find the number of overlay reads, procedure calls, and the load address of all procedures.

AND EVEN MORE!

DISK CACHE can be incorporated in your program to speed up disk reads for data bases, overlays, et al. Multi-file full screen BROWSE works on any text file. Pascal ENCRYPTOR makes your source safe from prying eyes, improves compile speed 15-30% and leaves the code 100% functional. SHELL generator creates fast compiling shells of unexercised code.

Two DSDD disks with complete Source Code, 100 page printed manual, 30 day guarantee!
Requires Turbo Pascal 3.0 and DOS 2.X or 3.X. Runs on IBM PC/XT/AT and compatibles. Call for generic MSDOS support.

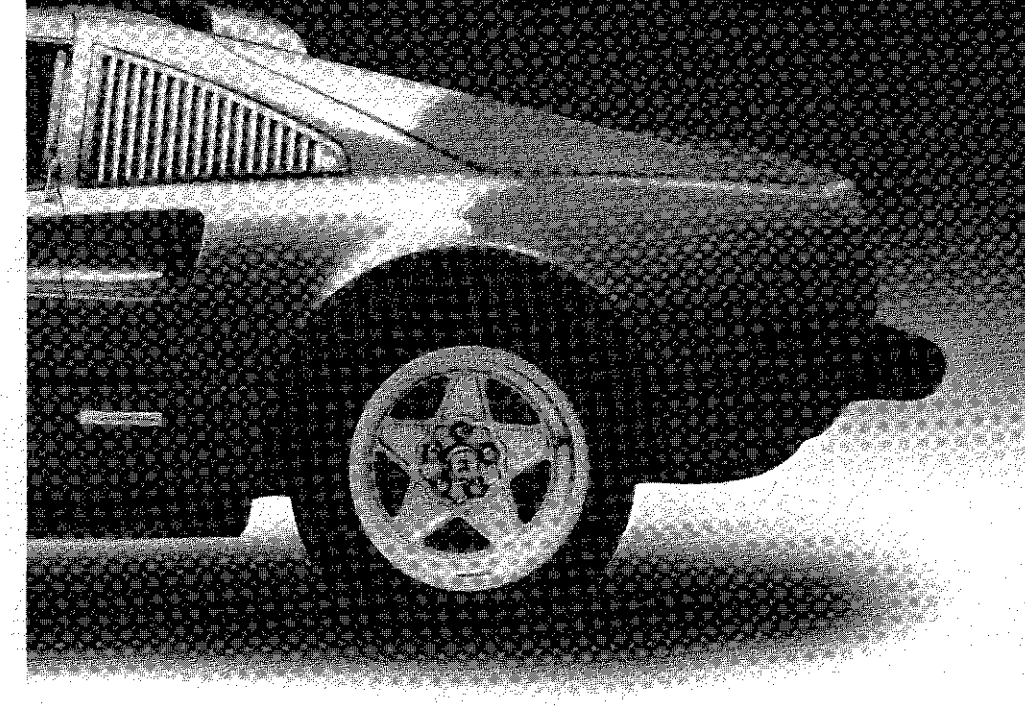
$85 complete

Also get the TurboPower Utilities with the acclaimed Pascal Structure Analyzer
Includes a Pretty Printer, Execution Profiler, and powerful Text and Command Automation Tools. With full source $95, executable only $55.

Credit Card Orders only call Toll-free 7 days per week
(US)800-538-8157x830 (CA)800-672-3470x830
PO, COD, Dealers, Questions, Brochures, call or write:

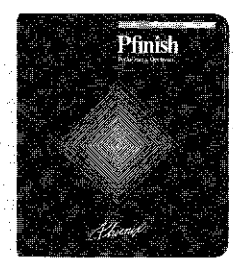
TURBO Power
478 W. Hamilton #196
Campbell, CA 95008
ph. 408-378-3672
M-F 9AM-5PM PST

Circle **25** on the reader service card



Why Send Anything But Your Best Software Into Final Production?

You don't have to with Pfinish™! You'll find all the inefficient parts of any MS-DOS® or PC DOS program. Assembler. C. PASCAL. FORTRAN. BASIC. Get comprehensive reports that show you the number of times various sections of code in a program are executed. Down to the instruction level. Displayed. Stored on disk. Or printed out. In tabular form or histograms.



Unlike other analyzers which use arbitrary address ranges, Pfinish can generate reports which reflect the structure of your program.

Using symbol table information provided by either our Plink™86 linkage editor or Microsoft®'s LINK, Pfinish divides your program into functional subroutines so you can check the number of times each subroutine is entered, as well as the number of instructions executed while within the subroutine. In addition, Pfinish can monitor any address range or interrupt inside or outside your program.

You can do a dynamic program scan. Pick out the most frequently executed subroutines. Track interrupts. Handle overlays. Find a programming error that costs your program valuable time. Rank program subroutines from the most to the least frequently executed. And, you can analyze any memory-resident program in your system since Pfinish can be set up to stay permanently resident.

This Pfinish report shows, for each routine, the number and percentage of its calls, the number and percentage of its instructions, and the clock time it consumed:

HIT COUNT	HIT %	RUN COUNT	RUN %	TICK COUNT	ADDRESS
79280	96	396400	91	92	CXA38
2570	2	10280	7	5	CXC33
2540	1	7620	1	25	_SBRK
84390		414300		122	

Pfinish is priced at $395 and is available for 8086-based micros running MS-DOS.

Find out more about Pfinish today! Call (800) 344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix Computer Products Corp., 320 Norwood Park South, Norwood, MA 02062

PROGRAMMERS' PFANTASIES™ BY

Phoenix

Pfinish and Programmers' Pfantasies are trademarks of Phoenix Computer Products Corporation.
Plink86 is a trademark of Phoenix Software Associates Ltd. MS-DOS and Microsoft are registered trademarks of Microsoft Corporation.

Circle **18** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

·hard disk problems·
·the no-backup blues
·sniffing the wind
.. AND MORE!!!

Excerpts from DTACK Grounded:
The 68000/32081 Systems Journal

And a Response to Peter Norton's Letter about Compilers in the Last Issue of *PJ*

by FNE, © 1985 by Digital Acoustics, Inc.

THOSE PC/AT DISK PROBLEMS:

Many of you know that there are about 20 or 30 rumors floating around explaining why the AT hard disks are so unreliable, as well as a few more IBM-sponsored rumors asserting that all is well with the AT hard disks.

We think there is more than one problem. In fact, we see THREE problems. One is that cheap hard disks are often sensitive to mechanical shock and stress after being formatted. What kind of shock and stress? The kind you get when you pick the AT up to move it across the desk or such. This torques the hard disk, and when you set the AT back down, the head alignment doesn't match the formatted tracks anymore.

This one is actually pretty obvious. And the torquing might be done at night by the maid conscientiously lifting one corner of the AT high in the air to dust thoroughly under the unit, hmmm? When the XT came out a few years back, it had this problem too, but folks forget.

Problem number two is DOS 3.0, which is highly reliable except when it has to work with faster hard disks than the really slow units used in the XT. And the PC/AT has much faster hard disks than the XT. DOS 3.1 is supposed to fix this problem. We forget whether we read about this in *PC WEEK* or *InfoWorld*, but it makes sense and explains why folks have been pointing fingers at DOS 3.0 for some time now, but without being specific.

Problem number three is that there is a logic error in one of the Western Digital chips used in IBM's hard disk controller card. Control Data Corp. found out about this logic error some time back, so when you upgrade a floppy-disk AT using a CDC drive, you get a replacement chip which has the error fixed. CDC has been very quiet about this fix, but *Electronic News* blew the story a couple of weeks ago, which is where we learned about it.

So wait for DOS 3.1, buy a floppy-based AT, and upgrade it with a CDC disk, then make damn sure nobody jerks the AT around on your desk (such as the maid lifting the chassis by one corner to dust under it at night), and you have a perfectly good AT — as soon as you buy a streaming tape to back the hard disk up, that is.

THEM NO-BACKUP BLUES:

Pete S. tells us about a colleague at his workplace, in a large West Coast aerospace firm's research labs, who recently purchased an AT and feverishly recorded data on the big hard disk for a period of six weeks, at which time the hard disk went BLOOEY! Bye, bye, six weeks of data. IBM offered sympathy and a new replacement hard disk.

It is hard to believe, but it continues to be a fact that 90% of the folks who own IBM personal computers with hard disks DO NOT BACK UP THEIR HARD DISKS! Not EVER! Incredible!

A COOPERATIVE DESIGN:

The first generation of 64K DRAMs simply did not work very well. This is true regardless of the manufacturer or country of origin. Intel's original 64K DRAM design failed so miserably that Intel, the inventor of dynamic RAMs, dropped out of the market for several years and has only now reentered the market with a vastly overpriced product whose only virtue is lowered power consumption.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Program Optimization Services Make BASIC Programs Run Like Assembly!

Software Studios, Inc. announces POS, Program Optimization Services, which will convert your BASIC or BASICA source code into an optimized .COM file to make your screen displays print four to ten times faster giving you: pop-up screens; word processing speed (like Personal Editor); instant display of telecommunications data; quicker program loading speeds; smaller program files. Makes BASIC programs look like Assembly!

POS is not a program. POS is a service. You send your source code and receive back a compiled .EXE or .COM file that is guaranteed to execute AT LEAST 30% faster! If you subsequently make program modifications, only the changes need to be optimized.

For further information, contact:

8516 Sugarbush • Annandale, VA 22003 • (703) 978-2339

Circle **24** on the reader service card

In Japan, industry overseer MITI decided that the thing to do was to pool the design talent and expertise of the leading five companies to produce a reliable and producible second generation design. So that is what happened, and is why the DTACK Grande can use any one of five Japanese 64K DRAMs interchangeably but can't use T.I. chips (pin 1 refresh) or Motorola chips (different timing). (Such a competitive endeavor is illegal in the U.S. because of anti-trust laws.) Once the design was completed, the cooperative was broken up and the Japanese began to compete. Believe it or not, *that also meant competing with each other!*

What is the number one objective of a Japanese company? Why, maximizing sales! So each Japanese company INDEPENDENTLY lowered prices of their 64K DRAMs early in the product cycle to hasten the design changeover from the 16K part. Profits were not just secondary but TERTIARY, as is usual Japanese business practice. The Japanese 64K DRAM suppliers were all too successful in their mutual but independent objectives: Mostek, T.I., and Motorola executives started screaming to their senators about unfair competition, known as "dumping" in the vernacular. "Dumping" is when a company sells export parts for less than a fair price.

So U.S. politicians jawboned their Japanese equivalents over these "dumping" charges. The Japanese, ever sensitive to good relations with trading partner U.S. and also wishing to avoid more stringent LEGAL export restrictions, asked these five Japanese companies to cut their exports of 64K DRAMs — and the same thing went for the (then) new 2K x 8 static RAMs. The five companies, as always working cooperatively with their helpful government, DID cut exports.

The way they cut exports — all five of them — was to RAISE PRICES *and let diminished demand reduce shipments to the U.S.*! This probably produced MORE profits for the companies than the larger shipments at the (previously) lower profits. That would enormously please an American company, but remember, the primary business objectives of Japanese companies are different.

DTACK SNIFFS THE WIND:

This simultaneous, voluntary price increase happened early in 1982. Along in here we got wind of what had happened, but first let us continue with the official U.S. reaction to this turn of events:

The U.S. Department of Justice noticed that five Japanese companies had simultaneously raised prices of their static and dynamic RAMs. They immediately jumped to the conclusion that the companies had put their heads together and conspired to fix prices and restrain trade. (That is illegal in the U.S. under anti-trust law and ALSO illegal for foreign companies exporting to the U.S.) This Justice Dept. investigation got under way while the U.S. Department of Commerce was still investigating those SAME five companies for "dumping"! We still have the lengthy press clippings from *Electronic News* (Aug 2 & 9, '82) describing this situation. Let us repeat this: The five companies were simultaneously being investigated by one U.S. agency (Commerce) for selling below cost and by another U.S. agency (Justice) for conspiring to set high prices! Some of the Japanese involved are probably still shaking their heads over those crazy Americans!

(RE)ENTER DTACK GROUNDED:

Just before the Justice Dept. got involved, we had already discovered that something strange was floating in the wind. We were paying about $9 for our 100nsec Toshiba 2K x 8 static RAMs, but a W. German reader had written to us about $4.50 prices for the same part in Europe. And *EN* had already published an article about $4 prices in Japan and $8 prices in the U.S. (in larger quantities than we were buying). So we called Toshiba headquarters in the U.S. to ask, "what gives?" We were told that there was Toshiba Japan, Toshiba Europe, and Toshiba U.S. and that each had "pricing autonomy"! Shortly thereafter, we ran into a brand-new, highly inexperienced salesman for one of the local authorized Toshiba distributors and inadvertently discovered that that distributor paid Toshiba $6 each for the parts they were selling to us for $9 each. Very shortly thereafter, without any change in the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

price Toshiba was charging its distributor, we were paying $4.75 each for the same part from the same manufacturer!

It was not just the (relatively low volume) static RAMs which were being sold for about double the Japanese or European prices, but the high-volume 64Ks were being sold in the U.S. in the same way and for the same reason — to comply with the "voluntary" export quota which had been imposed — well — requested by the Japanese government in response to criticism from the U.S., meaning Mostek, T.I., and Motorola.

ENTER THE TRADING COMPANIES:

A number of folks quickly learned that it was absolutely legal and highly profitable to fly to Japan with a bank draft and purchase, openly in the legal marketplace, Japanese memory chips, fly back to the U.S. with the memory chips, fully declare them at customs, pay the moderate customs fee, and then sell those memory chips on the open U.S. market. This proved to be a good deal for the trading companies, a good deal for their customers (Digital Acoustics, for instance), and a good deal for the Japanese companies whose products they purchased in Japan. *Best of all from the Japanese point of view, those parts did not count as being exported to the U.S.!*

Some folks have called this a "gray" market. We disagree because no portion of any of these transactions is even remotely illegal. It is, however, a CASH market as opposed to the CREDIT market which prevails in the U.S. Since we had earlier instituted the practice of charging cash or C.O.D. for our DTACK customers (at a time when acoustic noise monitors sold on CREDIT paid the bills for Digital Acoustics), this proved to be no problem. Buying and selling on cash terms continues to be our normal order of business, to our considerable advantage in many cases.

For instance, we deliberately pay our wave-soldering supplier cash on the barrelhead at the instant he delivers the completed (but untested) DTACK boards. Since his other customers pay normal "30 day credit" terms (i.e., 45 to 150 days), our work gets priority. We are breaking in a new printed circuit board supplier on the same basis, and we are already getting priority.

Excuse the digression, we were discussing the two-level pricing of Japanese memory chips. YOU MAY NOT BELIEVE WHAT COMES NEXT:

Eventually Motorola, and especially T.I., got their 64K DRAM production lines up and running and achieving good yield. They were also profitable, especially because they only had to compete in this country with Japanese chips whose prices were artificially inflated to assure lowered sales (to keep within the quota). However, an American firm with very good yields, and one whose principal manufacturing plant for DRAMs was located in JAPAN (!!!!), would naturally look for larger markets. And so T.I., with a Japanese manufacturing plant, came to cast an avaricious gaze on the Japanese market for DRAMs! (Folks, we are not making this up!)

T.I. SWIVES U.S. FIRMS:

T.I., observing that there was a two-tier pricing scheme already in place, ADJUSTED ITS PRICING TO COMPLY WITH THOSE TWO TIERS! Which meant that T.I.'s U.S. customers for 64K DRAMs subsidized T.I.'s Japanese customers for 64K DRAMs! We will pause briefly to allow those U.S. citizens who bought T.I. DRAMs in recent years to discharge their expletives . . . .

As other U.S. companies ramped up their 64K DRAM production, they followed T.I.'s lead and adopted that two-tier pricing scheme as their own (why rock a profitable boat?). One company which did this with enormous success was a "little" company in Utah named "Micron Technology." 64K DRAMs were almost the only product Micron made, and it made them with DEVASTATING profitability. In recent financial reporting quarters, Micron has actually outpaced LOTUS (software, as in 1-2-3) in profitability and for a while reported after-tax profits of around 30% (!) of sales, meaning BEFORE-TAX profits of around 60% of sales (!!!).

Generic Word Processor

Developing a Word Processor/Text Editor? Why reinvent the wheel? Software Studios has developed a generic word processor written in BASICA with the following features:

Top of File	Margin Setting	Block Copy
Bottom of File	Clear	Block Move
Line Insert	Paragraph Reformat	Block Delete
Line Delete	Center Text	Data Encrypt/ Decrypt
Save File	Page Mark	
Load File	Search/Find	File Merge
Print File	Block Mark	Quit

In addition, the following function keys are operative: Home, End, Page Up/Down, Cursor Up/Down/Right/Left, Insert, Delete. All commands are shown on screen under the Function Keys or Alternate Function Keys. Includes TURBOSCREEN subroutine that paints screen twice as fast as BASIC in either interpretive or compiled.

You get commented source code on disk in BASICA with user manual, installation instructions, compilation instructions, and user tutorial.

Demo Disk $10 **Source Code $765**

8516 Sugarbush • Annandale, VA 22003 • (703) 978-2339

Circle **24** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For a variety of reasons, Micron decided to throw sand in the gears of international commerce. In the fall of 1984, when the official high-quantity price for 64K DRAMs was nearly $4, and the "gray" market price was nearly $3, Micron announced that it was taking large-quantity orders for delivery in the fourth quarter of 1984 for 64Ks at $1.85 (200nsec) or $1.95 (150nsec). *This caused a sensation in the U.S. and is widely credited with precipitating the massive plunge in DRAM prices which then took place.*

THE FALL OF THE SECOND TIER

It is not commonly known that all Micron did was drop its U.S. prices to parity with its Japanese prices and thus bring an end to the two-tier pricing system that had become entrenched because almost everyone (as in suppliers) benefited. Micron did NOT drop its Japanese prices!

As a direct result of the end of the entirely artificial two-tier pricing scheme for memory chips, YOU can save a LOT of money when buying large-memory computers (and attached processors) from those manufacturers who respond quickly to price changes. (AHEM!) If YOU paid a lot more for memory a while back than you would have if you had waited to buy now, the main cause was that DAMN TWO-TIER PRICING SCHEME!

Let us underline the point: If you bought a large-memory computer in the past three years, but not in recent months, a totally artificial and highly unfair two-tier pricing scheme cost YOU money — probably lots of it. If you enjoyed buying a large-memory computer in recent months at drastically lower prices, it was the lower prices due to the demise of the two-tier scheme which were responsible for your good fortune.

And now you may have an intuitive understanding for a point which you may have overlooked or considered too abstract: The export quotas on Japanese cars have cost the American consumer an ENORMOUS amount of money. We have seen the figure of $800,000 placed on the annual cost of saving ONE job for a UAW (United Auto Worker) union member. You see, absent low-priced competition from Japanese cars, the Big Three U.S. auto companies jacked up their prices and profits accordingly. And that almost completes our lesson in international trade and economics for this issue.

This is the capper: The time is now ripe for Mostek, Motorola, and T.I. to bitterly complain to U.S. officials that the Japanese are unfairly "dumping" 256 K DRAMS in the U.S., and HERE WE GO AGAIN, FOLKS!

Response to Peter Norton's thoughts about compilers in the last issue of PJ

I almost regret terminating the newsletter *DTACK Grounded* since my opinions have of late been faulted by a much better grade of critic. I do wish my critics would pay more attention to what I wrote, though.

Peter Norton in his letter states that he:

1. Wrote those utilities in the language "C", which some computer professionals do not consider to be a high level language at all.

2. Went to the vast effort of writing his own I/O routines rather than using the included utilities. I thought a compiler was supposed to take the workload off the programmer!

3. Modified the "prologue-epilog" routines the compiler uses. Am I to believe that this is what ALL compiler-users do?

4. Specially selected a "C" compiler that made the above possible. How many high level language programmers have the knowledge and ability to do all of the above? Yet Peter did the above *"without too much sweat"?* (Emphasis added)

Peter did all this work *for a utility which will be used less than 0.1% of the time?* This is the same person who asserts that *I* am a fanatical adherent to efficiency?

If Peter is willing to go to all that effort for a utility package that will be used very infrequently, I wonder how much effort he would approve expending for a mass-market operating system, interactive BASIC, spreadsheet, and word processor — all of these being software that will be used a high percentage of the operating time of the PC system?

Is it possible that Peter has overlooked the fact that the IBM PC, which is rumored to be a somewhat successful piece of hardware in the mass market, has an operating system (MS-DOS, aka PC-DOS), an interactive BASIC (GW BASIC, aka BASICA), and a spreadsheet (1-2-3) that are all written in assembly? And that the successful word processors appear to be written in assembly as well?

Let us for a moment consider the advantages and disadvantages of writing commonly used mass-market software in assembly or HLLs:

Writing mass-market software in an HLL has the advantages of less programming effort over a shorter period of time (meaning lesser-skilled, hence cheaper, programmers can be used), transportability, and easier maintenance. All of these advantages accrue to the programmer or to his/her employer. The disadvantages of HLL-based programs are their slowness and the larger amount of RAM consumed. Both of these disadvantages accrue to the user, the guy or gal with the checkbook.

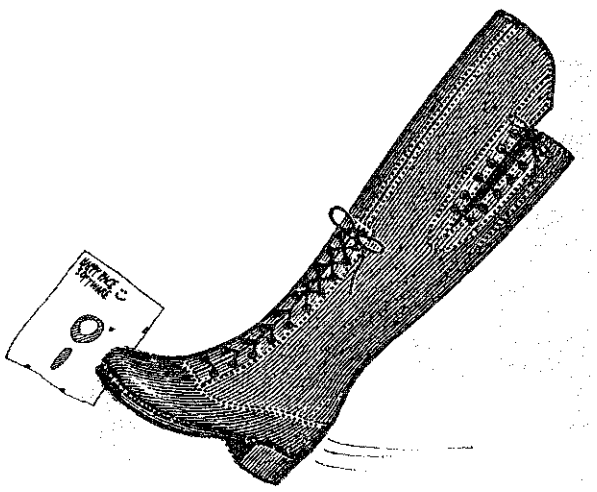
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Writing mass-market software in assembler has the disadvantages of a longer development time, requiring more skilled (hence, more expensive) programmers, untransportability, and more difficult maintenance. All of these disadvantages accrue to the programmer or to his/her employer. The advantages of mass-market software written in assembly are speed and a smaller amount of RAM consumed, leaving more for the user's programs and data. Both of these advantages accrue to the user, the guy or gal with the checkbook.

Can it surprise ANYONE that catering to the customers' wants and needs is often highly profitable? Can it be a surprise, then, that the vendor of assembly-based MS-DOS and GW BASIC has just broken ground on FOUR high-rise buildings in formerly sleepy Bellevue? Can it be a surprise that Lotus collected over $150 million for its assembly-based 1-2-3 in the past 12 months alone?

However, I am now directing Digital Acoustics into the mass-market software field (an operating system and BASIC for 68000-based machines such as the new Atari 520ST), and I have had a change of heart. I now recommend to other software houses — who are now my competitors — that they write all of their software in a high level language, especially 68000-based operating systems and interpretive BASICs!

—FNE □



Booting A Program. Method #2.

PRODUCTIVITY UTILITIES

From Opt - Tech Data Processing

NEW **On-Line Help** NEW

A comprehensive utility for adding help windows to your programs. It provides an efficient method for storing help texts in libraries and standard routines for interfacing your programs with these libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its position on the screen and the display colors used.

On-Line Help can be interfaced with interpreted Basic and all popular compilers.

On-Line Help includes a Help Library Manager, Library Interface Routines and source code examples. **$149.**

Opt-Tech Sort

High performance sort/merge utility can be used as a stand-alone program or called as a subroutine to many languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and Dbase. Up to nine sort control fields (ascending or descending), all common data types supported. Output file can be full records, keys or pointers.

Written in assembly language for high performance. Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$99.**

Scroll & Recall

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs are fully documented and are compatible with the IBM-PC, XT, AT and compatible machines using the DOS operating system.

Visa, M/C, Check, Money Order, COD, or PO accepted.
Quantity and Dealer discounts available.

To order or to receive additional information write or call:

Opt-Tech Data Processing
P.O. Box 678 — Zephyr Cove, NV 89448
(702) 588-3737

Circle **16** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AUTOINC EXPLAINED: EVALUATING THE AUTO-INCREMENT WITH RESPECT TO ADDRESSING AND DEREFERENCING

by Randall B. Swan

I've put this article together from many years of teaching C at New York University, from our C seminar workshops, and from numerous projects coded in C.

AUTOINC is a reference explanation of all possible auto-incrementing with dereferencing (taking the value of a pointer with an asterisk). Prefix and postfix auto-increment will behave significantly differently. Using AUTOINC as a look-up chart, one ought to be able to understand any possibility. Certain constructs have too many side effects and should be avoided for clarity, if nothing else.

Randall B. Swan Copyright © The C-Team, Inc.

When the prefix auto-increment ++ and the indirection * occur in the same expression, their precedence is the same. They bind right to left; that is, they associate right to left.

The suffix (postfix) auto-increment ++ does not concern precedence in practice because it always occurs after the expression is evaluated. Effectively, the indirection * always evaluates before the suffix auto-increment ++.

Auto-decrement: Operations apply in the same manner as in auto-increment.

The precedence table on page 49 of K&R (Kernighan and Ritchie's *The C Programming Language*) contains these rules.

The fundamental principle to remember is that the postfix auto-increment ALWAYS happens after the expression is evaluated.

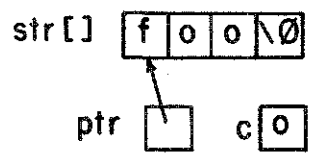
Dereferencing (indirection) is the process of taking the value at an address. This is done by putting an asterisk in front of a pointer name or any other name that means an address; i.e., array names. For instance:

```
char *p;
p = (char *) 0x100;
/* jam address 100 Hex into p */
putchar(*p);
```

This will output one ASCII character value obtained from location 100 Hex.

Assume these declarations for the examples that follow:

```
char c;
char str[] = "foo";
char *ptr = str;
```



The declaration

```
char str[] = "foo";
```

assumes a Null terminating byte, so the declaration is actually

```
char str[4] = "foo";
```

To initialize an array in this manner, these declarations should be external; i.e., above main() or outside any function. Alternatively, the declaration

```
static char str[] = "foo";
```

can be used inside main() or any other function. For main() this works very well. However, for other functions this automatic initialization occurs only the first time the function is called and is ignored on subsequent calls.

Randall Swan sheds a little light on some of the deeper aspects of C, including Autoinc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

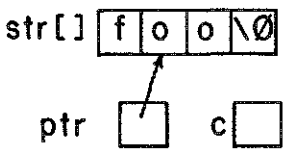
EXAMPLE 1

```
c = *++ptr;
c = *(++ptr);
```

Both of these are exactly the same. The expansion is

```
ptr = ptr + 1;
c = *ptr;
```

Ptr increments to the next address and then obtains the value at that address for c. c contains the first 'o' from "foo".



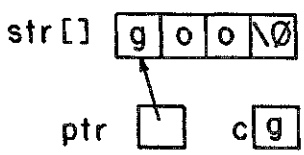
EXAMPLE 2

```
c = ++*ptr;
c = ++(*ptr);
```

Both of these are exactly the same also. The expansion is

```
*ptr = *ptr + 1;
c = *ptr;
```

Obtain the value at the address in ptr and add one to it. The 'f' of "foo" becomes a 'g'. c contains the new value 'g', and the value at the address in ptr has also changed to a 'g'in the array str[ ].



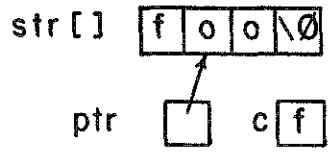
EXAMPLE 3

```
c = *ptr++;
```

The expansion is

```
c = *ptr;
ptr = ptr + 1;
```

Obtain the value at the address in ptr and place it in c. Then increment ptr to the next address. c contains the 'f' of "foo", and ptr ends up looking at the address of the first 'o' of "foo".



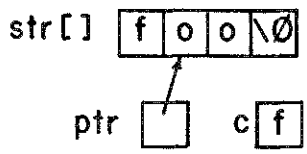
EXAMPLE 4

```
c = *(ptr++);
```

The expansion is

```
c = *(ptr);
ptr = ptr + 1;
```

Here we have a real surprise. It is exactly the same as *ptr++ in Example 3. The postfix auto-increment always happens after the expression is evaluated whether in parentheses or not. The parentheses are misleading. Only prefix auto-increment has precedence.



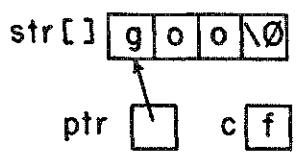
EXAMPLE 5

```
c = (*ptr)++;
```

The expansion is

```
c = (*ptr);
*ptr = *ptr + 1;
```

Obtain the value at the address in ptr and place it in c. c contains the 'f' of "foo". Then increment the value at the address in ptr so that the data in the string str[ ] changes to "goo".



EXAMPLE 6

```
c = (++*)ptr;
```

This will generate a "syntax error." It does not mean anything and won't compile.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

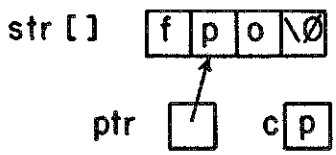
EXAMPLE 7

```
c = ++*++ptr;
c = ++*(++ptr);
c = ++(*++ptr);
c = ++(*(++ptr));
```

All of these are exactly the same: They associate right to left for evaluation regardless of the parentheses. The expansion is

```
ptr = ptr + 1; /* inner ++ */
*ptr = *ptr + 1; /* outer ++ */
c =*ptr;
```

Advance the address in ptr by one so that we're looking at the first 'o' of "foo". Then obtain that value and add one to it, changing the 'o' to 'p'. c contains the character 'p', and the data in str[ ] has also changed to "fpo".



EXAMPLE 8

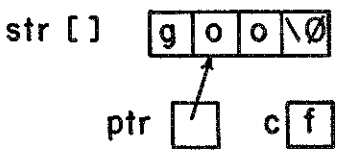
/swan 25/

```
c = (*ptr++)++;
```

The expansion is

```
c = (*ptr);
*ptr = *ptr + 1;     /* outer ++   */
ptr = ptr + 1;       /* inner ++   */
```

Obtain the value at address ptr and place it in c so that c contains an 'f'. Then the value at ptr is auto-incremented with the outer ++ because of the parentheses around (*ptr), so 'f' becomes 'g'. Then the address of ptr is auto- incremented by the inner ++. Thus str[ ] is changed to "goo", and ptr ends up looking at the first 'o' in "goo".



EXAMPLE 9

```
c = *(ptr++)++;
```

This generates an "illegal lvalue" error. The attempted expansion would be

```
c = *(ptr);          /* assignment first */
(ptr++)++;           /* expand both ++'s */
(ptr++) = (ptr++) + 1;   /* outer ++     */
(ptr=ptr+1) = (ptr=ptr+1) + 1;
```

The grouping of parentheses shows the expansion. The leftmost ptr+1 is not a legal left side destination (lvalue).

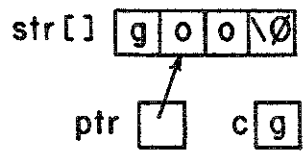
EXAMPLE 10

```
c = ++*ptr++;
```

The expansion is

```
*ptr = *ptr + 1;     /* prefix ++       */
c = *ptr;
ptr = ptr + 1;       /* suffix ++       */
```

Obtain the value at the address in ptr. Add one to the value and place it in c. c contains a 'g'. The data in str[ ] also has been changed to a 'g'. Then the address in ptr is advanced by one and sits looking at the first 'o' in str[ ].



EXAMPLE 11

```
c = ++(++*ptr);
```

This generates an "illegal lvalue" error. The attempted expansion would be

```
++(*ptr = *ptr + 1)      /* inner ++          */
(*ptr = *ptr + 1)  =  (*ptr = *ptr + 1)  +  1;
```

The leftmost *ptr + 1 is not a legal left side destination (lvalue).

□

Readers can reach Randall B. Swan at
60 East 12th Street, New York, NY 10003, (212) 477-0253.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C POWER

FUNCTION LIBRARIES
C POWER WINDOWS
SUPERFONTS FOR C
B-TREE LIBRARY
ISAM DRIVER
PC UNIX
C-TERP *INTERPRETER FOR C*
PC LOCATE *ROM any EXE file*

FREE! UNIX for the PC! FREE!

(with any purchase of $250 or more before 2/24/86)

FUNCTION LIBRARY

6 LIBRARIES
325 FUNCTIONS
FULLY TESTED
SUPERB DOCUMENTATION
BETTER FUNCTIONS

Most complete screen handling plus graphics; cursor/keyboard/data entry; string handling including word wrap; status and control; utility/DOS BIOS/time/date functions; printer control & more. BEST you can get Functions you NEED but don't have! Full source code. No royalties. Member: Entelekon's C Power family. 4 disks, 400 page manual $129.95

C POWER WINDOWS

MOST POWERFUL YET
POP-UP WINDOWS FOR
Menus/Overlays
Help Screens
Messages/Alarms
ZAP ON/OFF SCREEN
FILE-WINDOW MANAGEMENT
COMPLETE CONTROL OF:
Cursor
Attributes
Borders
AUTOMATIC
Horizontal & Vertical Scrolling
Word Wrap
Line Insertion

The most powerful, flexible and easy to use windowing package available! Many types of menus. Highlighting. Move data between files, keyboard, program and windows. Status lines. Change size/location/overlapping. Full source code. No royalties. Member: Entelekon's C Power family. 3 disks, 225 page manual $129.95

SUPERFONTS FOR C

SUPER SIZE CHARACTERS
Monochrome adapter
Color/graphics adapter
8 FONT LIBRARIES

Dramatic high impact screens/titles/messages. Use with or without windows. Great for projection TV. Create own font & image libraries or use ours. Use with windows for special effects or animation. Full source code. No royalties. Member: Entelekon's C Power family. Font and Function Library $49.95

B-TREE LIBRARY

POWERFUL DATA MANAGER
Fixed length records
Variable length records
FAST
EASY TO USE
16.7 MILLION RECORDS/FILE
16.7 MILLION KEYS/FILE

Fast B-tree indices. Add/remove keys Find first/last/next/any key. Find keys by Boolean selection. Read/write/delete or add records to file. PC-DOS, UNIX, XENIX, CP/M. Full source code. No royalties. Member: Entelekon's C Power family. 1 disk and manual $79.95

ISAM DRIVER

Makes programming with B-Tree Library easier, faster and simpler. Full source/no royalties $49.95

C-TERP

INTERPRETER FOR C
NO COMPROMISE, FULL K&R
BUILT IN SCREEN EDITOR
FAST, FAST COMPILE/LINK
USE EXISTING LIBRARIES!
SYMBOLIC DEBUGGING
SINGLE STEPPING
RAVE REVIEWS!

Powerful editor. Breathtakingly fast compile/link. Edit to run in 1 second. Automatic positioning in editor when error detected. Set breakpoints, execute C expressions directly. Supports 8087, batch mode, large model, external libraries, runtime tracing. Excellent manual. By Gimpel Software. "You couldn't ask for a more productive capability." See C Journal, Volume 1 Number 2. Member: Entelekon's C Power family. 2 disks and manual $299.95

C UTILITIES

MAKE UTILITY: *Powerful! Save time, automatically update your programs, libraries & documents. Avoid errors. $59.95*

LINT UTILITY: *Finds errors, saves time, full K&R, easy to use $129.95*

ROM UTILITY: *Creates ROM-able code for ANY .exe file $199.95*

OPERATING SYSTEMS

O/S TOOLBOX: *Build your own from this* *O/S function library. Multi- user, multi-tasking, event driven, networking. Source code $99.95*

PC MVS: *If you like MVS on a VAX, you'll* *love PC MVS on your PC! Multi-user, multi-tasking, networking. Includes source code! $99.95*

PC UNIX: *Multi-tasking, networking multi-* *user UNIX. Ideal for developing software! With source code! $99.95*

COMBINE AND SAVE!

C LIBRARY plus C WINDOWS
BOTH for only $179.95
+ SUPERFONTS FOR C $199.95
(A $310 VALUE)

C BUSINESS LIBRARY
INCLUDES C FUNCTION LIBRARY, C POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM
ALL for $299.95
(A $440 VALUE.....plus $100 PC UNIX free)

C PRODUCTIVITY LIBRARY
INCLUDES C FUNCTION LIBRARY, C POWER WINDOWS, SUPERFONTS FOR C, and C-TERP INTERPRETER
ALL for $459.95
(A $610 VALUE.....plus $100 PC UNIX free)

C TOTAL LIBRARY
INCLUDES C FUNCTION LIBRARY, C POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM and C-TERP C interpreter
All for $549.95
(A $740 VALUE...plus $100 PC UNIX free)

THE C POWER FAMILY ... YOUR COMPLETE SOURCE FOR C

Entelekon

12118 Kimberley, Houston, TX 77024 713-468-4412 VISA-MASTERCARD-CHECK-COD

Circle 9 on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SORTING USING AN AVL TREE

by Robert Ackerman

The necessity for preparing a 60,000-word sorted list with unique words and all inflections and derivations of any given word for a spelling checker led to the implementation of this sorting algorithm in C. When I started this task, I had no sorting tool. Initially I thought that the SORT utility provided with PC-DOS might be of some use to me. A couple of trials with lists of words showed me that it was hopelessly inadequate: A 2,000-word list took about twenty minutes to sort. I then started casting about for an adequate tool. A reference in *The C Programming Language* by Kernighan and Ritchie turned me towards sorts based on trees. After a perusal of *Sorting and Searching, The Art of Computer Programming, Vol. 3* by Donald A. Knuth, I decided that tree sorts seemed the best choice. In particular, the type described first in 1962 by two Russian mathematicians, G. M. Adel'son-Vel'skii and E. M. Landis, looked like a good candidate.

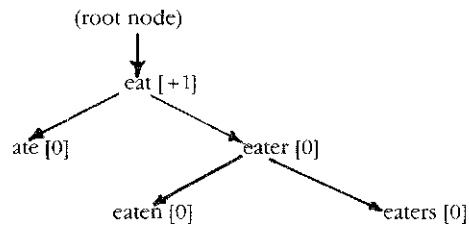
It seemed appropriate because I was starting with a base list of root words already in sorted order. To those root words, however, needed to be added inflections (tense and number) and affixes (both prefixes and suffixes). Thus a great deal of the list was maintained in sorted order, with occasional radical departures from the order. For example, the following might be a part of the list:

eat
eats
ate
eaten
eater
eaters

AVL TREES

To create a sorted list, you need to be able to find an item in the list with a given sorting key, and you need to be able to insert an item in the list at that point. Some sorting algorithms are efficient at one or the other of these operations, but usually not both. Tree representations of linear lists are efficient at finding an item having a certain sorting key and inserting an item at a specified place. The Adel'son-Vel'skii-Landis (AVL) method represents a linear list as a tree, which takes advantage of both those efficiencies. In addition, an AVL tree is a balanced tree. This means that if any one subtree starts to become longer than any other, the tree is reorganized to maintain the same depth among all the subtrees. In a tree sort with an unbalanced tree, nearly sorted items create a tree with one long branch. After N items have been inserted, inserting item N + 1 requires N comparisons. (Nearly sorted lists are also the worst case for other sorting algorithms: for instance, the Quicksort algorithm.) By contrast, insertion in an AVL tree never requires more than lg N + c comparisons, where c is a small constant. Sorting a list of N words thus takes on the order of N lg N steps.

A sorted tree consists of nodes such that the left child is always smaller than the parent, and the right child is always larger. For instance,

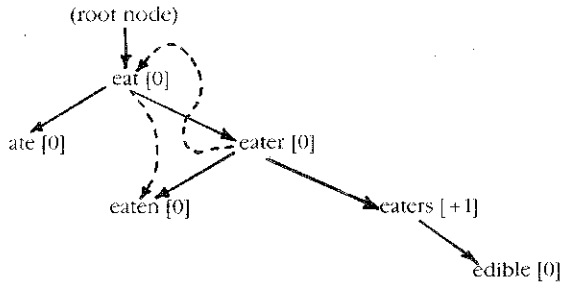


The search and insertion algorithms for maintaining this order are relatively straightforward. The numbers in brackets are the balance factors at each node. A plus indicates that the right subtree is longer than the left, a minus that the left is longer than the right, and a zero indicates that the subtrees are the same length. In an AVL, or balanced, tree no subtree is more than one level deeper than any other, so the balance factors at the nodes can only be −1, 0 or +1.

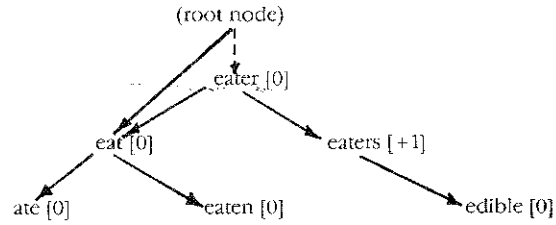
Balancing and rotation are a bit more convoluted. For instance, consider adding the word *edible* to the above tree. The program first finds the location where the new word should be inserted, in this case as the right child of *eaters*. As the program searches, it marks a node where rebalancing may need to begin. This node always has a balance factor of +1 or −1, and is selected only if the node to be added falls in the longer subtree; that is, the added node may increase the balance factor to +2 or −2. In this case, it marks *eat*. The program also marks the parent of that node (*root*, in this case). (If the program were adding *autolysis* and *eat* were the start of a subtree, it would mark *eat*, but it would not rebalance there because the addition would be on the shorter, rather than the longer, subtree.) After inserting the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

new node, the program readjusts the balance factors from the marked node on (shown below), and reconnects the links of the tree to maintain the new balance (shown as dashed lines below).



Finally, the program connects the parent of the node marked for rebalancing to its new child (shown as the dashed line below).

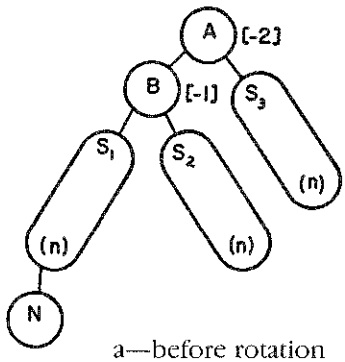


A double rotation proceeds in exactly the same fashion, except that three instead of two nodes need to be relinked, requiring that five instead of three pointers be reset. That, however, is the maximum when any word is inserted. Randomly ordered lists require rebalancing and rotation only about one-third of the time. The worst case is a sorted list, in which case insertions cause alternate single and double rotations.

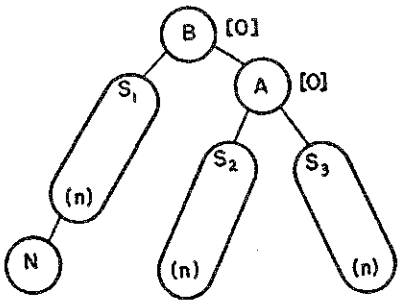
The two general cases for any tree (or subtree) are shown below.

Si is a subtree of length (n) or (n+1), and N is a newly added node. Before adding N, the tree had the AVL property.

1 — SINGLE ROTATION

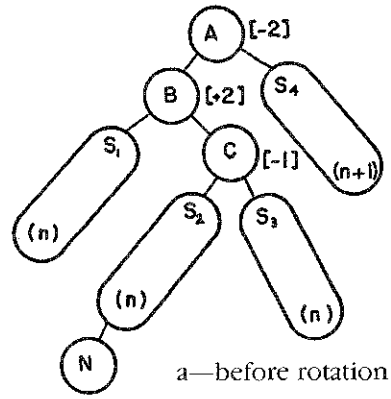


a—before rotation

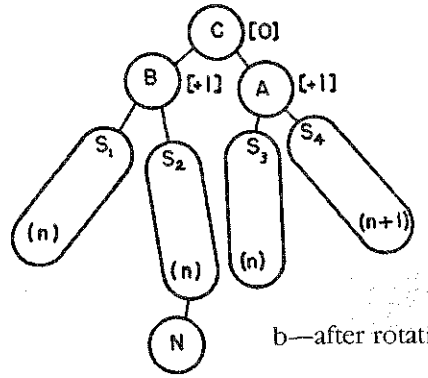


b—after rotation

2 — DOUBLE ROTATION



a—before rotation



b—after rotation

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INTRODUCING DATALIGHT C

The Datalight C Compiler for MSDOS is a full C with all K&R constructs, including bitfields, plus the version 7 extensions. Other features of the compiler are:

$60

- Produces object files (.obj) so just the MSDOS linker is required.
- Floating point performed with 8087 or automatic software floating.
- Over 100 compact library functions with source.
- Compatible with the Lattice C compiler.
- Runs on IBM-PC, and compatibles, running MSDOS 2.0 or later.
- Complete, easy-to-read users' manual with index.
- Highly optimized code for production quality programs.

Datalight

11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

Outside USA add $10 shipping. Washington State residents add 7.9% sales tax. VISA and MasterCard accepted.

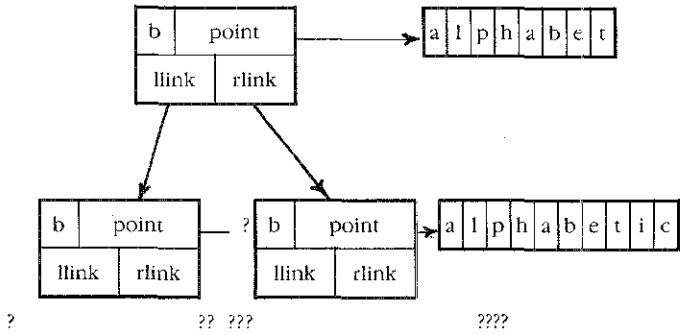
IBM-PC, a trademark of IBM; MSDOS, a trademark of Microsoft Corp.; Lattice C, a trademark of Lattice Corp.

Circle 7 on the reader service card

IMPLEMENTATION CONSIDERATIONS

For an AVL sort to be more efficient than other sorts, the number of items to be sorted must be large. In my case, I wanted to sort as many of the 60,000 words at one time as possible; in practice, this turned out to be about 7,000+ words at a time. The lists had to be sorted several times, as mistakes, omissions, and misspellings were discovered. (In practice, several lists covering parts of the alphabet were maintained, and ultimately merged with duplicates discarded to make the final master list.)

In addition, AVL trees are best suited to internal sorts. Sorting English words presents a problem in this respect. The minimum length of words to be considered was two letters; the maximum encountered was 25 letters. The average length, however, is about nine letters. Any fixed length structure wasted most of the space allocated to it, thus eating up memory available for more items, and requiring more parts to sort. These constraints led to the choice of data structures and memory allocation methods, and the compiler choice. The data structure, which consists of one integer and three pointers, comes from the discussion in Kernighan and Ritchie, except that the integer is put to a different use. The integer contains the balance factor at that node of the tree. Two pointers point respectively to the left child and the right child of that node. The third pointer points to an array containing the word at that node. A representative node might be depicted thus:



With this structure, an array exactly long enough for each word can be allocated.

Memory is allocated for each word as it is put into a structure, thus ensuring that only the amount of memory necessary for the list at hand is allocated, and the amount of memory allocated for each word is exactly large enough to contain that word. The most efficient sorting technique — that is, the one that necessitated maintaining the fewest separate files of words — required a C compiler that uses all available memory for data: one of the "large data" model compilers, which can use up to 1M of data space. The combination of data structure and memory model determines the maximum number of items that can be sorted at one time. These compiler models use four-byte pointers (including the NULL pointer, which must be defined as 0L). This gives a total size of 13 bytes for each node. Because each word takes on average 10 bytes (9 for the letters plus 1 for the null character), the total space required for each word is about 23 bytes. The memory allocation methods on some compilers add another 4 bytes for each structure allocated, bringing the total up to 31 bytes per word when those compilers are used. Thus, on a computer with 512K of RAM, this program could sort between 15,000 and 20,000+ words at one time, depending on the choice of compiler.

The choice of a "large data" model compiler also has some effect on the speed of the sort because the program has to process four-byte rather than two-byte pointers. Applications that could contain their data in 64K of data space could achieve some speed advantage over the "large data" models. I have done no comparisons, however, between either large and small models or this algorithm and others. The running time of this implementation seems to be bound more by disk access time than processing time, although that is strictly impressionistic.

The program below is based on Knuth's Algorithm A (pages 455 and following). *The Design and Analysis of Computer Algorithms* by Aho, Hopcroft and Ullman also gives a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

description of AVL trees and an analysis of AVL trees, and another algorithm for sorting using AVL trees can be found in *Introduction to Computer Organization and Data Structures* by Harold S. Stone. Knuth's, though, seemed to be the clearest and most straightforward. This implementation differs from Knuth's in one minor respect: The search algorithm is implemented in a recursive function. This takes some memory in the stack, and since the implementation has no boundary checks as it as written at present, the stack could overwrite data as well. The memory lost due to recursion is trivial since 20,000 words would require a maximum of 15 copies—perhaps 750 bytes. Overrunning the heap is a more serious problem; the trivial conversion of the search function to make it nonrecursive would obviate much of this potential hazard. Another solution is more careful memory management.

The recursive function for printing (or writing) the tree also presents a similar problem, although it puts fewer parameters and automatics on the stack with each call. Rewriting this function as a non-recursive function, however, would sacrifice its marvelous simplicity and elegance.

To minimize disk activity, the program reads each word, allocates a structure for it, and links that structure initially in an unordered, singly linked list before it begins any sorting, rather than reading a word, allocating a structure for it, and then immediately inserting that structure in its sorted order. Even though I had careful control of the data, I knew that the program would occasionally encounter duplicate words. From the above list again, *ate* might be entered in its strictly alphabetic order and it might be entered again as an inflected form of *eat*. In cases like this, the duplicates were discarded and the structures returned to the free memory pool. Other applications—for instance, those that count occurrences of a given word in a document, where there are certain to be multiple occurrences—would need a different method for reading and inserting words.

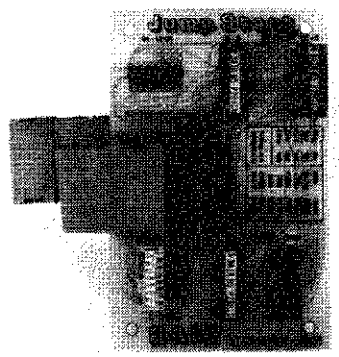
JUMP START Boots IBM Faster for $79.95

CUTS BOOT TIME
Reduces 640K system boot time to 5 seconds with memory test disabled or 10 seconds enabled (DIP switch options).

PAYS FOR ITSELF QUICKLY
Time savings for $30,000/year employee who reboots 20 times per day recovers entire cost in 3 weeks.

TAKES NO SLOTS
IBM ROM plugs into Jump Start and board replaces ROM in socket. Original BIOS ROM accessible via DIP switch option.

for PC1, PC2, or XT



power window
FREE
14 Day Trial
Check It Out
Jump Start can save RAM Disks.

JS/PC1
for original 16K PC.
JS/PC2
for 64K PC and 16K PC with hard disk via BIOS change.
JS/XT
for XT and Portable.

DAIR Computer Systems
3440 Kenneth Drive Palo Alto, CA 94303
415 - 494 - 7081

Circle 5 on the reader service card

The word list I developed needed to contain only lowercase alphabetic characters. Thus a strictly ASCII collation order, rather than a dictionary collation order that disregards capitalization, was acceptable. For this reason, the program also contains a very simple string comparison function rather than a library function. This permitted as well the use of register variables in the function used most often throughout the program. In addition, one compiler's library function collated improperly. It collated something before nothing, resulting in the following collation sequence, for instance:

eaten
eaters
eater
eats
eat

Other sorting tasks, of course, would require other comparison functions.

Famous Computer Personalities #54: Estbrooke, Dennis III, Americas first carriage return character.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

①

```
/*
 ---------------------------------------------------------------------

   Declarations, definitions for avl.c
     (Memory conserving version, 17 September 85)

 ---------------------------------------------------------------------
*/

#define USE_MSG "Usage = programname inputfile outputfile"
#define NO_IN   "Can't find input file"
#define NO_OUT  "No output file specified"
#define DISKPROB        "Problems writing to disk."
#define NOMEM   "Out of memory. Too bad, ha ha."

/* #define   NULL    0L;     NULL pointer definition for D model */
            /* #ifdef LONG in stdio.h; -dLONG option in compiler */

char        *getword();  /* function returning a string pointer */
char        *allocation function();     /* memory allocator */

char            *w;         /* control variable & pointer to word */

int             len = 0;                /* word length */
int             memflag = 1;    /* enough memory? */
int             writeflag = 0;  /* problems writing to disk? */
   /* prints message, calls shutdown() and _exits */
int             quit();
int             shutdown();     /* closes files */
   /* finds predecessor node and inserts new node */
int             search();
int             rot_bal();      /* rotates and balances tree */
   /* recursive inorder tree traversal; prints tree */
int             treeprint();
int             strcmp();           /* string comparison function */

struct  tnode {     /* basic node; K & R, p. 131 + Knuth, p.457 */
                char    *point; /* pointer to word */
                int     b;                      /* balance factor */
                struct  tnode   *llink; /* pointer to left child */
                struct  tnode   *rlink;/* pointer to right child */
                };

struct  tnode   *list,*current,*root,*rebal,*parent;/* node ptrs */

struct  tnode   *talloc(); /* memory allocator; returns node ptr */

FILE            *infp, *outfp;
FILE            *fopen();
/*
 ---------------------------------------------------------------------

  Reads a file of words, one to a line, into a singly-linked
    list and then inserts them in sorted order in an AVL tree.
    Discards duplicates.

  The practical limitation on the number of items to be sorted
    depends on amount of memory available, and the memory
    allocation method. Several functions are recursive.

  After the list is sorted, it is written to a file, using a
    recursive inorder tree traversal.

  The program takes two inputs on the command line:

                        input file name
                        output file name

  The program should be compiled using a compiler that supports
    long data models: 64K program space and 1M data space.
    This requires that NULL be defined as a long pointer.

    Note that this program includes the code for strcmp() to
    collate strictly by ASCII, not dictionary, order.

  Robert Ackerman, 13 March 1985
    revised, 19 March 1985
    revised, 17 April 1985 (read into singly-linked list)
    revised, 17 September 1985 (conserve memory)

 ---------------------------------------------------------------------
*/
```

②

```
/* inclusions, definitions, declarations */

#include        "stdio.h"
#include        "avl.h"

main(argc,argv)
int             argc;
char            *argv[];
{

/* open files, check for validity, print error messages */

        if (argc < 3) {
                printf("%s\n",USE_MSG);
                return;
        }
        if ((infp = fopen(argv[1],"r")) == (FILE *)NULL) {
                printf("%s\n",NO_IN);
                return;
        }
        if ((outfp = fopen(argv[2],"w")) == (FILE *)NULL) {
                printf("%s\n",NO_OUT);
                return;
        }
/*
 ---------------------------------------------------------------------
   initialize tree
 ---------------------------------------------------------------------
*/
                { /* get the header node */
        if ((root = talloc()) == (struct tnode *)NULL)
                quit(NOMEM);
        }
              /* right link never used in root node */
        root->rlink = (struct tnode *)NULL;
        root->b = 0;                    /* tree height */
        root->point = (char *)NULL;  /* never used in root node */

/*
 ---------------------------------------------------------------------
  read words from input file into tree nodes; link nodes together
   as a singly-linked list
 ---------------------------------------------------------------------
*/

        list = root;    /* use root as list header temporarily */
        while ((w = getword(word)) != (char *)NULL) {
                if (writeflag != 0) {
                /* getword() sets writeflag if probs */
                        quit(DISKPROB);
                }
                if ((list->llink = talloc()) == (struct tnode *)NULL) {
                /* get node */
                        quit(NOMEM);
                }
                else {
                        list = list->llink;     /* advance list pointer */
                        list->rlink = (struct tnode *)NULL;
                        /* set right link to NULL */
                        list->b = 0;    /* set balance factor to zero */

                        /*
                         ---------------------------------------------
                           allocate structure for word = length of word + 1
                           list points to the current node; point points to the
                            current structure to receive a word
                         ---------------------------------------------
                        */

                        len = strlen(w);
                        if ((list->point = allocation function(len +1)) ==

                                (struct tnode *)NULL) {
                                quit(NOMEM);
                        }
                        strcpy(list->point,w);
                        /* copy w into structure at point */
                }
        }
        list->llink = (struct tnode *)NULL;     /* mark last node */
```

Instant-C™:
The *Fastest* Interpreter for C

Runs your programs 50 to 500 times faster than any other C language interpreter.

Any C interpreter can save you compile and link time when developing your programs. But only *Instant-C* saves your time by running your program at compiled-code speed.

Fastest Development. A program that runs in one second when compiled with an optimizing compiler runs in two or three seconds with *Instant-C*. Other interpreters will run the same program in two minutes. Or even ten minutes. Don't trade slow compiling and linking for slow testing and debugging. *Only Instant-C will let you edit, test, and debug at the fastest possible speeds.*

Fastest Testing. *Instant-C* immediately executes any C expression, statement, or function call, and display the results. Learn C, or test your programs faster than ever before.

Fastest Debugging. *Instant-C* gives you the best source-level debugger for C. Single-step by source statement, or set any number of conditional breakpoints throughout your program. Errors always show the source statements involved. Errors always show the source statements involved. Once you find the problem, test the correction in seconds.

Fastest Programming. *Instant-C* can directly generate executable files, supports full K & R standard C, comes with complete library source, and works under PC-DOS, MS-DOS, or CP/M-86. *Instant-C gives you working, well-tested programs faster than any other programming tool.* Satisfaction guaranteed, or your money back in first 31 days. *Instant-C* is $495.

Rational
Systems, Inc.

P.O. Box 480
Natick, MA 01760
(617) 653-6194

Circle **20** on the reader service card

Moving?

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal*. Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 3000
Dept. EE
Denville, NJ 07834

GROWING OLD?

...waiting for C programs to compile and link?

Use C-terp
the **complete** C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, etc. etc. For the ultimate in customization, editor source is available for a slight additional charge of $98.00.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Object Module Support– Access functions and externals in object modules produced by your compiler. **New:** We are now supporting **Microsoft** 3.0, **Mark Williams** & **Aztec C** in addition to C.I.C86 & Lattice.
- Complete Multiple Module Support.
- Symbolic Debugging-- Set breakpoints, single-step, and directly execute C expressions.
- Many more features including batch mode and 8087 support.
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation & shipping within U.S. PA residents add 6% sales tax. Specify compiler.

- C-terp runs on the IBM PC (or compatible) under DOS 2.x with a suggested minimum of 256Kb of memory. It can use all the memory available.

GIMPEL SOFTWARE
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 (Computer Innovations), Lattice (Lattice Inc.) IBM (IBM Corp.), C-terp (Gimpel Software), Microsoft (Microsoft), Aztec (Manx)

Circle **11** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tools for the Programmer from Blaise Computing

Save Up To $130 On These Special Offers!

TOOLS & TOOLS 2

For C or Pascal

For a limited time, pick up both packages and save $50 off our regular list price. The C version comes with libraries for the Lattice, Computer Innovations and Microsoft (version 2.03 and 3.00) compilers. The Pascal version supports IBM and Microsoft Pascal. **$175.**

VIEW MANAGER With Source

All libraries are included. Please specify C or Pascal. Regular $425. Save $130. **$295**

Blaise Computing provides a broad range of fine programming tools for Pascal and C programmers, with libraries designed and engineered for the serious software developer. You get clearly written code that's fully commented so that it can serve both as a model and also be easily modified to grow with your changing needs. Our packages are shipped to you complete with comprehensive manuals, sample programs and source code. None of the programs are copy-protected.

FOR C AND PASCAL PROGRAMMERS:

TOOLS ◇ $125
Extensive string and screen handling, graphics interface and easy creation of program interfaces. Includes all source code.

TOOLS 2 ◇ $100
Memory management, general program control and DOS file support. Interrupt service routine support. Includes all source code.

VIEW MANAGER ◇ $275
General screen management. Create data entry screens that can be easily manipulated from your application program. Block mode data entry and retrieval with fast screen access.

VIEW LIBRARY Source ◇ $150
Source code to the VIEW MANAGER library functions.

ASYNCH MANAGER ◇ $175
Powerful asynchronous communications library providing interrupt driven support for the COM ports. All source code included.

FOR THE TURBO PASCAL PROGRAMMER:

Turbo POWER TOOLS ◇ $99.95
Extensive string support, extended screen and window management, interrupt service routines, program control and memory management, interrupt filters. All source code included.

Turbo ASYNCH ◇ $99.95
Interrupt driven asynchronous communication support callable from Turbo Pascal. ASYNCH is written in assembler and Turbo Pascal with all source code included.

PACKAGES FOR ALL PROGRAMMERS:

EXEC ◇ $95
Program chaining executive. Chain one program from another even if the programs are in different languages. Common data area can be specified. Source code included if you're a registered C TOOLS and C TOOLS 2 user.

SPARKY ◇ $75
Run-time resident (or stand-alone) scientific, fully programmable, reverse polish notation calculator. No limit on stack size, variables or tape. Includes all standard scientific functions and different base arithmetic.

TO ORDER, call Blaise Computing Inc. at (415) 540-5441

♦ 2034 Blake Street ♦ Berkeley, CA 94704 ♦ (415) 540-5441 ♦

BLAISE COMPUTING INC.

Circle 2 on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMBINING BINARY TREES

by John G. Mowat

A binary tree is commonly used to organize a stream of data that arrives in random order. Examples of such data streams are college students arriving for registration, books being checked out of a library, or sales occurring at a large store. As the data arrives, a procedure searches the tree to see if that kind of data is already present or not. If not, the search terminates at the exact place in the tree where the data should be added. At any time the data in the tree can easily be read out in sorted order.

Routines for working with binary trees are usually recursive; i.e., they call themselves and, accordingly, give the programmer a wonderful feeling of getting something for nothing — a great deal gets accomplished with a small amount of code. Also, binary trees are a dynamic data structure. This means that one does not know how much data there will be in the tree at the start of a program run. Instead of setting aside a portion of memory in the beginning as with ordinary program variables, the program allocates space in memory as needed. References 1 and 2 (listed at the end of this article) discuss binary trees. The following assumes your familiarity with some of the material in those references.

The general idea of this article is simple, but useful nonetheless: If a binary tree is saved to disk in a pseudo random order, it can be fed back into another binary tree (without destroying the balance of the tree — see below). Thus, two or more trees can be combined (merged). It turns out that in order to do this combining you need little more code than you would need in order to have a binary tree in the first place. The end result: A few simple binary tree routines replace separate routines for accumulating data, sorting it, and merging the sorted files. We will demonstrate such a system in this article with a short program written in Turbo Pascal.

Why would one want to combine binary trees? To answer this, let's consider the example of sales at a store. In fact, let's imagine that you own a large department store and want to do some sales analysis. At each computerized cash register, sales information is fed into a sequential file. The records in the sequential files are of TYPE shown in Figure 1.

```
TYPE
     DataType = RECORD
  key:Integer;
  qty:Integer;
  code:Integer;
END {DataType};
```

Figure 1. Data record for sequential files.

This record is somewhat idealized. In real practice the key would most likely be a string of characters and there might well be additional fields. In any case, the key identifies what was sold and provides access to a master file containing other attributes of the item sold; e.g., a key of 589 can access a large data file that might tell you that 589 = Desk lamp #1E42.3 with a price of $19.95, an inventory stock of 311, etc. The qty tells how many such items were sold, and the code stands as an example of other fields that might be in the record. The code might be equal to qty for a cash transaction and zero for a credit transaction. Then you could use the code to keep track of immediate cash flow generated by the transaction.

If you started a sequential file of these records twice a day at each cash register, you would shortly have a large number of such files available. Each file could be fed into a binary tree that would condense the data; e.g., if 37 desk lamps with key 589 were sold in 25 transactions, each of the 25 would take up a record in the sequential file, while in the binary tree these would all be combined into a single record for desk lamp 589. Each tree could be saved to disk in the way we will describe below, and you would have the basic data for a variety of sales comparisons. You could, for example, compare storewide sales for Tuesdays with those for Fridays, second floor with third floor, first week of the month with third week, or mornings with afternoons simply by combining the data from the appropriate trees; for example, to compare Tuesdays with Fridays, simply combine all the tree files that were generated on Tuesdays from all cash registers into a "big" tree and compare the output of this tree with another "big" tree made from the Friday data. In fact, as long as you keep the raw tree files, you can generate new kinds of comparisons, perhaps on the basis of ideas derived from seeing earlier ones.

Now that we see a possible application for the system, let's plunge into some of the details of the program listing. The trees themselves are constructed from records we call NodeType, which appear at the beginning of the program listing. The NodeType record is similar to the DataType record except for two additional pointer fields that point to other copies of the record (nodes), allowing the tree to be constructed dynamically in RAM as the data comes in from any source.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Circle 12 on the reader service card

The Fast Path for New Users of Turbo Pascal...

For TURBO PASCAL On The IBM PC

TRAINING WHEELS™

- A comprehensive step-by-step example to help you get started.
- 30 useful & necessary procedures—ready to be included in your program—each with source code and with a simple program illustrating its use: Display help screens & menus quickly, check the DOS environment strings, determine where the program was loaded from, manipulate the archive & read-only bits, determine the ROM date & type of machine & type of display being used, retrieve and parse the command line arguments, make your program preserve the color attributes set for DOS, move a file to a different directory without copying it—and many more.
- Helps get your application going quicker and provides examples of how the DOS function calls are used from Turbo.
- 80-page user-guide plus diskette.

Available for only $25.00
(plus $2 shipping). Cal. Res. add 7%.
Telephone Orders Welcomed (408) 733-4032

LANDMARK SOFTWARE

1142 Pomegranate Court
Sunnyvale, CA 94087

Going now to {Main} at the end of the listing, as we always do with a Pascal program when trying to get the big picture, we see that after such preliminaries as marking dynamic memory and initializing a tree, the program calls a procedure Do_Menu. Do_Menu puts up a menu (see Figure 2) and allows the user to repeatedly select menu choices.

```
Test Data Trees
24195

<1> Input from Keyboard
<2> Auto Input
<3> Show Tree, prefix order
<4> Show Tree, ordered
<5> Save Tree
<6> Combine Trees
<7> Erase Tree
<8> Quit

Enter: -
```

Figure 2. Program Menu.

Examining this menu we see a number, 24195, the number of 16-byte paragraphs remaining in dynamic memory where the tree is being built. Selection <1> allows you to add to the tree in memory from the keyboard. (Beware: The input routine is not user friendly.) Selection <2> generates a data stream using a random number generator. These selections let you build a tree starting from scratch. They simulate the creation of the trees from the sequential files mentioned above. Selection <3> displays the nodes of the tree in what is called prefix order (Reference 1). This is the order in which the tree is saved to disk. Selection <4> displays the nodes of the tree in sorted order. (To stop the display in selections <3> and <4>, toggle with Ctrl-s). In the display there will always be an artificial item 0 whose qty is the maximum depth of the tree. Selection <5> allows you to save a tree to disk. You are prompted for a filename, and the tree is saved. Selection <6> allows you to read a tree from disk into the tree in memory. Selection <7> wipes out the tree in memory by releasing dynamic memory back to the mark. Selection <8> terminates the program.

The heart of the entire process of constructing a tree in the first place, of adding to it, and of searching it is the procedure Enter_Item near the top of the listing. This procedure consists of an outer "shell," Enter_Item itself, and a "core" procedure Into_Tree, where the real work gets done. The reason for making two procedures instead of the single one as in References 1 and 2 is to save memory during recursive calls. Note that the "shell" has 5 parameters, the "core" only one. Every time a procedure is called, a copy of each parameter (or parameter address in the case of VAR parameters) is pushed on the stack/heap in dynamic memory. As there might be a large number of recursive calls, we save a great deal of space by introducing the recursive Into_Tree with only the one parameter it really needs. The shell Enter_Item carries the parameters that will be needed after Into_Tree locates or creates the proper node. If you look at the main line of Enter_Item, you see that all it does is initialize a counter, Cntr, to 0 and call Into_Tree. The variable Cntr keeps track of the length of the search path in the tree so that you can track the maximum depth of the tree and thus have some idea about its "balance."

Binary trees can be balanced, completely unbalanced, or partially balanced. See Figure 3.

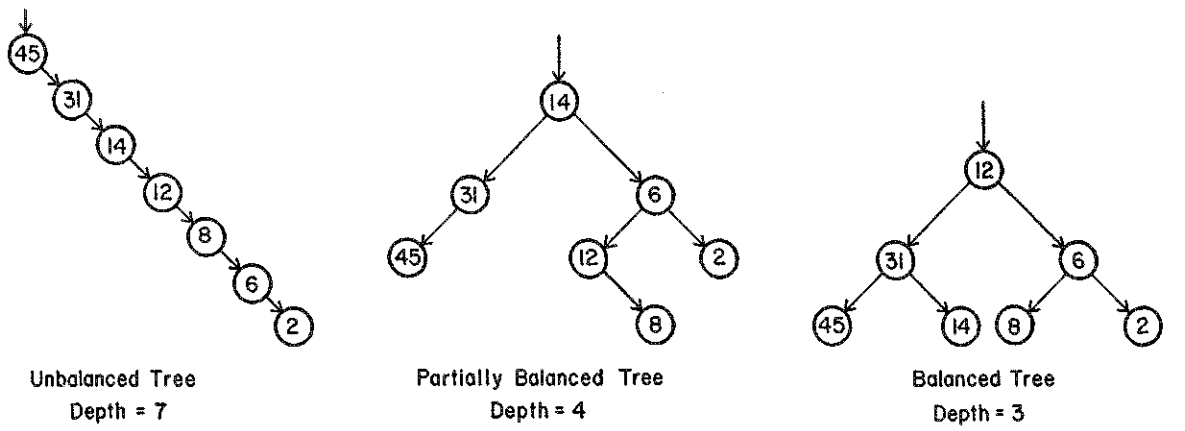
A balanced tree is one of minimum depth and shortest search path to any node. A completely unbalanced tree is equivalent to a linked list and has a single long search path. Balanced trees are "good," unbalanced ones "bad." Unbalanced trees are generated by ordered data, partially balanced, satisfactory trees by random data. In the scheme here, because the keys from the raw data will, in general, not be ordered, our strategy is to depend on this randomness rather than resort

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

to complicated code for balancing our trees. The variable Cntr allows us to see how well this strategy is working.

Procedure Into_Tree is a fairly standard search and insertion routine. Given Pt_Node, a pointer to a node, the procedure checks to see if the node really exists; if not (i.e., Pt_Node = nil), a new node is created and the input data is stored there (unless qty = 0). If the node exists, the key in the node is checked to see if it is greater than, less than, or equal to ky, the search key. If greater or less, Into_Tree calls itself, passing the left or right node pointer, and the process repeats until a pointer is nil or a matching node is found. If the keys are equal, the proper node is found, and the input data is added to what is already present at the node. You should note the apparently superfluous assignments to qy and cd. These assignments allow the VAR parameters qy and cd to return the proper values at the node, in case you are searching, or searching as well as updating the tree.

Figure 3. Binary Trees. Numbers are keys.



ED/ASM-86

The first truly integrated Editor, Assembler, Debugger, and Linker is now available from Oliver Computing Company.

*** The Editor supports block move, block copy, partial save and partial load insertion. All text is tokenized for both efficiency of storage and very fast assembly time. FAST screen I/O and separate editor/debug windows.**

*** The Assembler can be immediately invoked after any editting command. Output can be directed into memory, to a .COM file, .EXE, or to ED/ASM-86's own efficient .LNK file format. 8087/186/286 instructions supported. Macros and most standard pseudo-ops are supported.**
UNIQUE to ED/ASM is its structured assembly language capability. No more wasted labels and time! Multiple segmented programs can be be sent to .COM files.

*** The built in debugger supports symbolic disassembly, integration with the assembler, as well as disassembly of 8087/186/286 instructions. Display 8087 registers in scientific decimal format. SPLIT SCREEN debugging with a dynamic register window is easily invoked. Disassemble programs to disk file for easier analysis. "Single-step" data editting supported for bytes, words, double words, and 8087 floating constants in memory. You can "patch" assemble a line at a time with symbols from previous assembly (even define symbols as you go!)**

*** The linker supports ED/ASM-86 link files, which can be created from virtually any valid ED/ASM-86 source file, even lines with multiple external symbols in expressions. (Note: not compatible with usual link format)**

*** And many more features too extensive to be listed here.**

All of these features exist in one integrated program. A typical development cycle with ED/ASM-86 consists of entering a program with the editor, or editting an existing program, typing "A:,-IM" to assemble into memory, going into the debugger to immediately test the program. Then you go back to the editor for the next cycle.
ED/ASM was developed for Oliver Computing's own use, and has been used for over a year to enhance itself!

ED/ASM-86 is only $95. Send check or M.O. to:
Oliver Computing Company, P.O. Box 90140, Indianapolis, IN 46290 { (317) 849-4450 }
for immediate delivery via UPS.

Circle **15** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When passed a pointer to the root node at the top of the tree, the three procedures T__Dis__PrFix, T__Dis__Ord, and T__Put__Prfix all traverse the entire tree and put out the information at each node to the screen or to disk. All three are recursive in that each, when dealing with a node, calls itself with the left pointer, then the right pointer at the node. The routine T__Dis__Ord (Dis__Ord = display ordered) sends information to the screen between its recursive calls. The other two procedures (PrFix = prefix order) first send information to the screen or to disk and then do the two recursive calls. In the first case, the information comes out to the screen with the keys sorted in ascending order. This demonstrates the sorting aspect of binary trees, which would be used in the application when it was time to display the contents of "big," combined trees. With the PrFix procedures, information comes out with the keys in a scrambled order called prefix order in Reference 1. This ordering is what is needed for saving the tree to disk because it enables the tree to be recreated in memory without loss of balance. If the tree were saved with the keys in order, then when it was read back into a tree using Enter__Item, it would be completely unbalanced. As it is, the prefix-ordered file results in a tree with the same balance as when the tree was saved.

Other routines in the program carry out file handling and other housekeeping chores. They should be easy to understand from the listing. We hope that you have fun playing with this program and that it provides a useful basis for applications. For a 5-1/4" diskette (PC DOS format) with the text of this article and the program listing, send $9 to the address below. □

John G. Mowat
Mind Matters, Inc.
P.O. Box 758
Cottage Grove, OR 97424

References:

1. Wirth, Niklaus. 1976. *Algorithms + Data Structures = Programs.* Englewood Cliffs, N.J.: Prentice-Hall, 189-232.

2. Kernighan, Brian W. and Dennis M. Ritchie. 1978. *The C Programming Language.* Englewood Cliffs, N.J.: Prentice-Hall, 130-133.

PC TOOLS

AT LAST! Powerful disk utilities from the people who brought you Copy II PC.

Put all the popular features of the NORTON UTILITIES* together with a powerful DOS interface. Then make them ***resident*** like SIDEKICK** and what do you have? PC TOOLS—the only product that lets you execute virtually any DOS command from within any other running program! Here's how you might use it:

- **UNDELETE** an accidentally erased file while using your word processor!
- **FORMAT** a data disk without leaving your spreadsheet!
- **COPY** files from one subdirectory to another within your communications program!
- **LOOK** at another file (and even edit it) from your graphics package!
- **SEARCH** for any file name or text within a file without leaving your data base manager!
- **PRINT** a file (as text or hex and ASCII) at any time!

PC TOOLS works on most IBM compatibles, requires 192K, 256K to use the memory resident feature.

PC TOOLS is unprotected, of course, to work great from your hard disk!

CENTRAL POINT Software, Inc.
9700 SW Capitol Hwy., #100
Portland, OR 97219

503/244-5782
M-F, 8-5:30, W. Coast Time
VISA CHECK, COD WELCOME
(Prepayment Required)

$39.95
plus $3 s/h ($8 overseas)

*Norton Utilities is a registered trademark of Peter Norton **Sidekick is a registered trademark of Borland International

Circle 4 on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

①

```
        {18 Sep 85}
PROGRAM Exp_Tree;  {Explore Binary Trees}


 TYPE

  N_Ptr = ^NodeType;

  NodeType = RECORD
    key:        Integer;
    qty:        Integer;
    code:       Integer;
    pt_left:    N_Ptr;
    pt_right:   N_Ptr;
  END {Node};


 DataType = RECORD
   key:        Integer;
   qty:        Integer;
   code:       Integer;
 END {Data};

      VAR

        Pt_Heap_Mark :  ^Integer;
        Pt_Tree_Root :  ^NodeType;
        Data : DataType;
        Tfilename : String[12];
        Treefile : File of DataType;
        Outfile : Text;
        Depth: Integer;
        Max_Depth: Integer;
        ky, qy, cd: Integer;
        Time_to_Quit: Boolean;

    PROCEDURE Mk_rnd (VAR R: Real);

       BEGIN
         R := 4.000622 * ln (R + 3.141592654);
         R := Exp (R);
         R := R - Trunc (R);
       END;


    PROCEDURE Get_File_Name;
       BEGIN
         ClrScr;
         Gotoxy (5,5);Write('<Return> To Skip OR');
         Gotoxy (5,8);Write('Enter file name: ');
         Readln (Tfilename);
       END;

  PROCEDURE Enter_Item (VAR Pt_Node: N_Ptr; VAR Cntr: Integer;
       ky: Integer; VAR qy, cd: Integer);

    PROCEDURE Into_Tree (VAR Pt_Node: N_Ptr);


       BEGIN
         Cntr := Cntr + 1;

  IF

  (Pt_Node = nil) THEN BEGIN
       IF
```

②

```
    (qy <> 0) THEN BEGIN
           NEW (Pt_Node);
           WITH Pt_Node^ DO BEGIN
              key := ky;
              qty := qy;
              code := cd;
              Pt_left := nil;
              Pt_right := nil;
           END;
                  END ELSE
     {qy = 0}
           Cntr := Cntr - 1;
                     END ELSE IF
(Pt_Node^.key > ky)  THEN Into_Tree (Pt_Node^.Pt_Left)
                     ELSE IF
(Pt_Node^.key < ky)  THEN Into_Tree (Pt_Node^.Pt_Right)
                     ELSE
(Pt_Node^.key = ky)  WITH Pt_Node^ DO BEGIN
                              qty := qty + qy;
                              qy := qty;
                              code := code + cd;
                              cd := code;
                     END;
     END {Into_Tree};


    BEGIN {Enter_Item}
      Cntr := 0;
      Into_Tree (Pt_Node);
    END;

  PROCEDURE New_Max_Depth;
     BEGIN
       qy := Depth - Max_Depth;
       Max_Depth := Depth;
       cd := 0;
       Enter_Item (Pt_Tree_Root, Depth, 0, qy, cd);
     END;

 PROCEDURE Try_Open (VAR Result: Integer);

    BEGIN
    {$I-}
      Assign (Treefile, Tfilename);
      RESET (Treefile);
    {$I+}
      Result := IOresult;
      IF Result <> 0 THEN
        IF Result <> 2 THEN BEGIN
           Gotoxy (5,10);
           Write ('Something Screwed up.');
           Gotoxy (5,12);
           Write ('IO_Result is',Result:3);
           Time_to_Quit := True;
      END;
    END;

 PROCEDURE Try_Combine;

    VAR
      Result: Integer;
      ch:   Char;
      First_Read: Boolean;

    BEGIN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

③

```
    Try_Open (Result);
    IF NOT Time_to_Quit THEN BEGIN
       IF Result <> 0 THEN BEGIN
            Gotoxy (5,10);
            Write ('Couldn''t find that dang file anywhere');
            Read (Trm, ch);
                          END ELSE BEGIN
       {Open went OK}
          IF Depth = 0 THEN BEGIN
             qy := 1;
             cd := 0;
             Enter_Item (Pt_Tree_Root, Depth, 0, qy, cd);
          END;
          READ (Treefile, Data);
          WHILE NOT EOF (Treefile) DO BEGIN
               WITH Data DO BEGIN
                    ky := key;
                    qy :=  qty;
                    cd :=  code;
               END;
               Enter_Item (Pt_Tree_Root, Depth, ky, qy, cd);
               IF Depth > Max_Depth THEN
               New_Max_Depth;
               READ (Treefile, Data);
          END;
          CLOSE (Treefile);
       END;
    END;
  END;


PROCEDURE T_Dis_PrFix (Pt_Node: N_Ptr);

   BEGIN
     IF

  (Pt_Node <> nil) THEN BEGIN
            WITH Pt_Node^ DO Writeln ( key: 8, qty: 15, code: 15);
            T_Dis_PrFix (Pt_Node^.Pt_left);
            T_Dis_PrFix (Pt_Node^.Pt_right);
    END;
   END;


PROCEDURE T_Dis_Ord (VAR I: Integer; Pt_Node: N_Ptr);

   BEGIN
     IF

   (Pt_Node <> nil) THEN BEGIN
             T_Dis_Ord (I, Pt_Node^.Pt_left);
             I := I + 1;
             WITH Pt_Node^ DO Writeln (Outfile, I:9,'.', key: 8,
                                       qty: 15, code: 15);
             T_Dis_Ord (I, Pt_Node^.Pt_right);
    END;
   END;

PROCEDURE Store_Tree;

   PROCEDURE T_Put_PrFix (Pt_Node: N_Ptr);

      BEGIN
        IF
```

④

```
  (Pt_Node <> nil) THEN BEGIN
        IF Pt_Node^.qty <> 0 THEN BEGIN
             Data.key := Pt_Node^.key;
             Data.qty := Pt_Node^.qty;
             Data.code := Pt_Node^.code;
             WRITE (Treefile, Data);
        END;
        T_Put_PrFix (Pt_Node^.Pt_left);
        T_Put_PrFix (Pt_Node^.Pt_right);
      END;
   END;

  BEGIN
    qy := -Max_Depth;  {so node with depth won't be saved}
    Enter_Item (Pt_Tree_Root, Depth, 0, qy, cd);
    ASSIGN (Treefile, Tfilename);
    REWRITE (Treefile);
    T_Put_PrFix (Pt_Tree_Root);
    CLOSE (Treefile);
    qy :=  Max_Depth;  {restore maximum depth}
    Enter_Item (Pt_Tree_Root, Depth, 0, qy, cd);
  END;


PROCEDURE Feed_Tree;

   VAR
     Line,
     Item_Key:  Integer;
     Qty,
     Code:      Integer;
     Finished: Boolean;

   BEGIN
     ClrScr;
     Gotoxy (7 ,1 ); Write ('To quit enter "-1" for Item Key');
     Gotoxy (4 ,3 );
     Write (' Item Key          Quantity          Code');
     Line := 4;
     IF Depth = 0 THEN BEGIN
         qy := 1;
         cd := 0;
         Enter_Item (Pt_Tree_Root, Depth, 0, qy, cd);
     END;
     Finished := false;
     REPEAT
       IF (Line < 25) THEN line := line + 1;
       Gotoxy (7, line); Readln (Item_Key);
       IF Item_Key = -1 THEN Finished := True ELSE BEGIN
          Gotoxy (24, line); Readln (Qty);
          Gotoxy (38, line); Readln (Code);
          ky := Item_Key;
          qy :=  Qty;
          IF Code = 0 THEN cd := 0
                      ELSE cd :=  Qty;
          Gotoxy (38, line); Write (cd);
          Enter_Item (Pt_Tree_Root, Depth, ky, qy, cd);
          IF Depth > Max_Depth THEN New_Max_Depth;
       END;
     UNTIL Finished;
   END;

PROCEDURE Auto_Feed;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

⑤

```
   VAR
     I, Max, N: Integer;
     R: Real;
     Item_Key:  Integer;
     Qty,
     Code:      Integer;

   BEGIN
     ClrScr;
     IF Depth = 0 THEN BEGIN
           qy := 1;
           cd := 0;
           Enter_Item (Pt_Tree_Root,  Depth, 0, qy, cd);
     END;
     Gotoxy (5, 2); Write ('Give Max Key: ');
     Readln (Max);
     Gotoxy (5, 4); Write ('Give number of items: ');
     Readln (N);
     Gotoxy (5, 6); Write ('Give seed 0..1: ');
     Readln (R);
     Gotoxy (5, 9); Write ('This takes about 1 sec per 5 items.');
     FOR I := 1 TO N DO BEGIN
         Mk_rnd (R);
         Item_Key := Trunc (Max * R) + 1;
         Mk_rnd (R);
         Qty := Trunc (125 * R);
         IF  Qty MOD 7 < 2 THEN Code := 0
                           ELSE Code := Qty;
         ky := Item_Key;
         qy := Qty;
         cd :=  Code;
         Enter_Item (Pt_Tree_Root, Depth, ky, qy, cd);
         IF Depth > Max_Depth THEN New_Max_Depth;
     END;
   END;


PROCEDURE Do_Menu;

   VAR
     I:  Integer;
     ch: Char;
     Mem: Real;

   PROCEDURE DispPrfix;
      BEGIN
        ClrScr;
        IF Pt_Tree_Root = nil THEN BEGIN
             Gotoxy (10,12);
             Write ('No Data in Tree');
             Gotoxy (10, 15);
                                END ELSE BEGIN
             Gotoxy (4 ,0);
             Writeln ('  Item Key       Quantity          Code');
             Gotoxy (0, 2);
             T_Dis_PrFix (Pt_Tree_Root);
        END;
        Writeln ('<Space> to continue');
        Read (Trm, ch);
      END;

   PROCEDURE DispOrder;
     BEGIN
```

⑥

```
           ClrScr;
           IF Pt_Tree_Root = nil THEN BEGIN
                    Gotoxy (10, 12);
                    Write ('No Data in Tree');
                    Gotoxy (10, 15);
                                   END ELSE BEGIN
              {Tree contains data}
                    Gotoxy (5, 8);
                    Write ('<P>  To Printer: ');
                    Read (Trm, ch);
                    Writeln;
                    ClrScr;
                    Gotoxy (0, 0);
                    IF ch = 'p' THEN BEGIN
                          CLOSE (Outfile);
                          ASSIGN (Outfile,'LST:');
                          Rewrite (Outfile);
                    END;
Writeln (Outfile,'     Line no    Key          Quantity         Code');
                    Writeln (Outfile);
                    I := 0;
                    T_Dis_Ord (I, Pt_Tree_Root);
                    Close (Outfile);
                    ASSIGN (Outfile,'CON:');
                    Rewrite (Outfile);
           END;
           Write ('<Space> to Continue');
           Read (Trm, ch);
         END;


    BEGIN {Do_Menu}
      Time_to_Quit := false;
      REPEAT
        Mem := MemAvail;
        IF Mem < 0 THEN Mem := 65536.0 + Mem;
        ClrScr;
        Gotoxy (20,1 ); Write ('Test Data Trees');
        Gotoxy (24,2 ); Write (Mem:5:0);
        Gotoxy (8 ,4 ); Write ('<1>  Input From Keyboard');
        Gotoxy (8 ,6 ); Write ('<2>  Auto Input');
        Gotoxy (8 ,8 ); Write ('<3>  Show Tree, prefix order');
        Gotoxy (8 ,10); Write ('<4>  Show Tree, ordered');
        Gotoxy (8 ,12); Write ('<5>  Save Tree');
        Gotoxy (8 ,14); Write ('<6>  Combine Trees');
        Gotoxy (8 ,16); Write ('<7>  Erase Tree');
        Gotoxy (8 ,18); Write ('<8>  Quit');
        Gotoxy (10,21); Write ('Enter: ');
        REPEAT
          Gotoxy (16, 21);
          ClrEol;
          Read (Trm, ch);
        UNTIL ch IN ['1'..'8'];
        CASE ch OF

          '1':          Feed_Tree;

          '2':          Auto_Feed;

          '3':          DispPrfix;

          '4':          DispOrder;

          '5':           BEGIN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                    Get_File_Name;
                    IF Tfilename <> '' THEN Store_Tree;
                  END;

          '6':    BEGIN
                    Get_File_Name;
                    IF Tfilename <> '' THEN Try_Combine;
                  END;

          '7':    BEGIN
                    Release (Pt_Heap_Mark);
                    Mark (Pt_Heap_Mark);
                    Pt_Tree_Root := nil;
                    Depth := 0;
                    Max_Depth := 1;
                  END;

          '8':    Time_to_Quit := True;

        END;
      UNTIL Time_to_Quit;
    END {Do_Menu};


BEGIN {Main}
  Depth := 0;
  Max_Depth := 1;
  Mark (Pt_Heap_Mark);
  Pt_Tree_Root := nil;
  ASSIGN (Outfile,'CON:');
  Rewrite (Outfile);
  Do_Menu;
  Release (Pt_Heap_Mark);
END.
```


A FULL C COMPILER FOR $49.95

The Eco-C88 C compiler is setting a new standard for price and performance. Compare Eco-C88's performance to compilers costing up to 10 times as much:

	Seive	Fib	Defref	Matrix
Execute	12.1 sec.	43.1 sec.	13.7 sec.	21.3 sec.
Code Size	7782	7754	7772	9120
Compile-link	76 Sec.	77 Sec.	77 Sec.	92 Sec.

Eco-C88 Rel. 2.20, on IBM PC with 2 floppy disks, 256K. Benchmarks from Feb., 1985 **Computer Language.**

Eco-C88 includes:
* All operators and data types (except bit fields)
* Error messages in English with page numbers that reference the **C Programming Guide** – a real plus if you're just getting started in C.
* Over 170 library functions, including color and transcendentals
* New Library functions for treating memory as a file
* User-selectable ASM or OBJ output (no assembler required)
* 8087 support with 8087 sensed at *runtime*
* cc and "mini-make" for easy compiles (with source)
* Fast, efficient code for all IBM-PC, XT, AT and compatibles using MSDOS 2.1 or later.
* Complete user's manual

If ordered with the compiler, the C library source code (excluding transcendentals) is $10.00 and the ISAM file handler (as published in the **C Programmer's Library,** Que Corp) in OBJ format is an additional $15.00. Please add $4.00 for shipping and handling. To order, call or write:

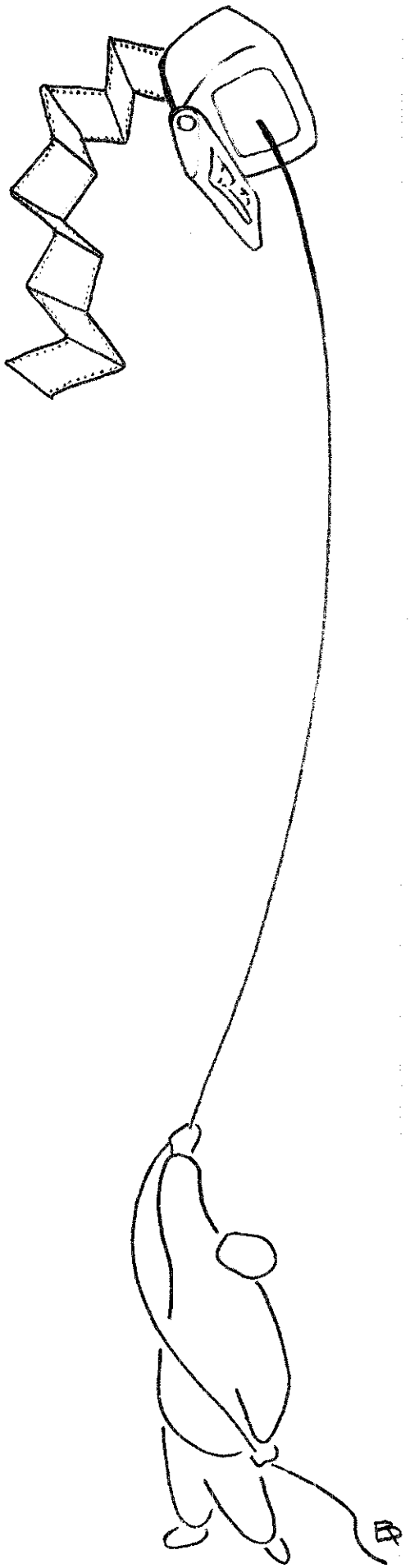
MasterCard VISA AMERICAN EXPRESS

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476 • 8:30-4:30

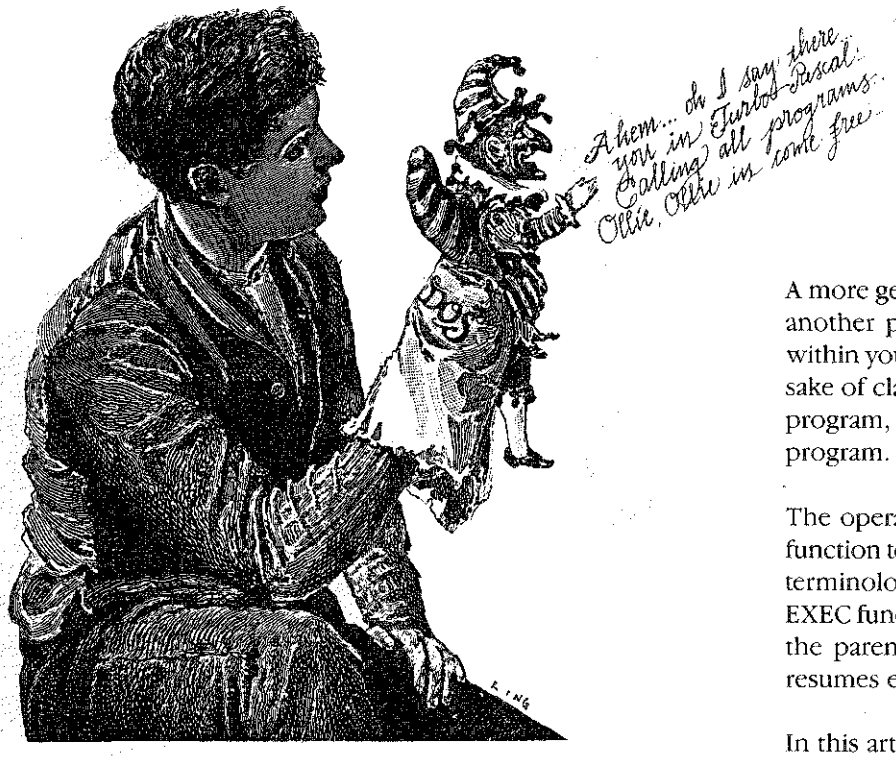
ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

Circle **8** on the reader service card



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



USING DOS TO CALL PROGRAMS FROM TURBO PASCAL

by Arvind Kumar

Have you ever wanted to call Lotus 123 to edit a spreadsheet from your Turbo Pascal application? Or to call WordStar to compose and edit some text from your Turbo Pascal program? Or to invoke CrossTalk in the middle of your program to connect to a remote machine?

If your machine runs MS DOS, all of this is possible, and more — if you understand a little bit about the operating system and its facilities. What you need is a way to run other programs from your Turbo Pascal program.

Turbo Pascal provides a built-in procedure, called Execute, to run other programs, such as the ones above, from your own Turbo Pascal program. However, the Execute procedure has some severe limitations. First, the program to be run must be in a .COM file format (no. EXE files allowed). Second, you cannot pass it a command line. Last, and most important, control passes to the other program irrevocably, which means that your own program is overwritten by the other program. When the other program finishes, control returns to the operating system.

A more general and powerful feature is the ability to FORK another program, where you run another program from within your own without terminating your program. For the sake of clarity, I will refer to your program as the "parent" program, and to the program it wants to run as the "child" program.

The operating system (hereinafter called DOS) provides a function to run a program from within another, or, using our terminology, to fork a child from a parent. This is the DOS EXEC function (hex 4b). Since DOS is not multitasking (yet), the parent remains suspended while the child runs and resumes execution when the child terminates.

In this article, I will develop two Turbo Pascal procedures based on the DOS EXEC function. The Fork procedure is capable of running any program, .COM or .EXE, and passing a command line to it, as in:

```
Fork('WS.COM', 'READ.ME');
```

The System procedure is more general and executes DOS internal commands as well as running programs, such as:

```
System('DIR *.TXT');
System('TYPE READ.ME');
```

The procedures themselves are modular and can be included in your Turbo Pascal applications with ease.

This article first develops some generic support routines for calling DOS functions and handling errors. It then goes on to develop the Fork and System functions. Each function is followed by discussion and numerous examples of its use. The article closes with a brief section on limitations and suggested enhancements.

In writing this article, I made no assumptions about your knowledge of DOS or of assembly language. Instead, I wrote it for Turbo Pascal enthusiasts and developers who do not have the time to delve into the complexities of assembly language but need to use advanced features of DOS. If you find the pace of the article a mite slow, you can jump directly to the listings and refer to the article when you need additional clarification.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CALLING DOS INTELLIGENTLY

To call DOS functions, Turbo Pascal provides the built-in procedure MsDos. To avoid any confusion between them, I will refer to the operating system as DOS (all upper case) and to the Turbo Pascal built-in procedure as MsDos (mixed case).

The MsDos procedure requires a Pascal record defining the registers of the Intel iAPX 8088 family of CPUs:

```
TYPE
 RegType =
   RECORD
     AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
   END;
```

To call DOS, load the required values into the fields of a variable of this type before passing it to the MsDos procedure. On return, these fields will be altered to reflect the results of the function call.

DOS indicates errors by turning on the carry flag. As a matter of good programming practice, we should check for errors after each call to the MsDos procedure.

Define a simple function, CallDOS, which passes a record of type RegType to the MsDos procedure and returns TRUE if the function call was successful and FALSE otherwise:

```
FUNCTION CallDOS(VAR Regs : RegType) : Boolean;

BEGIN { FUNCTION CallDOS }

MsDos(Regs);
CallDOS := (NOT Odd(Regs.Flags));

END;  { FUNCTION CallDOS }
```

The carry flag corresponds to the least significant bit in the Flags field of record Regs. On error, this bit is set, making the value of the Flags field odd. This is a convenient and effective way of checking for an error return from DOS.

Note also that the argument, Regs, is declared VAR so that return values from MsDos are available to the routine that calls CallDOS.

PRINTING ERROR MESSAGES, NOT NUMBERS

When a DOS function fails, the carry flag is set and the AX register contains an error code. This error code depends on the function called and the kind of error that occurred.

A good program should display helpful error messages, not cryptic error codes. With this goal in mind, write a function DOSErrorMsg to convert DOS error codes into messages in English. It could be called

```
WriteLn('Operation failed : ', DOSErrMsg(Regs.AX));
```

where Regs is a record of type RegType. The function is

```
TYPE
  AnyString = String[255];

FUNCTION DOSErrorMsg(AX : Integer) : AnyString;

VAR
  ErrNumStr : AnyString;

BEGIN { FUNCTION DOSErrorMsg }

CASE AX OF
  1  : DOSErrorMsg := 'Invalid function';
  2  : DOSErrorMsg := 'File not found';
  3  : DOSErrorMsg := 'Path not found';
  4  : DOSErrorMsg := 'Too many open files';
  5  : DOSErrorMsg := 'Access denied';
  7  : DOSErrorMsg := 'Arena trashed';
  8  : DOSErrorMsg := 'Not enough memory';
  9  : DOSErrorMsg := 'Invalid block';
  10 : DOSErrorMsg := 'Bad environment';
  11 : DOSErrorMsg := 'Bad format';
  ELSE
   BEGIN
   Str(AX, ErrNumStr);
   DOSErrorMsg := 'DOS Error ' + ErrNumStr;
   END;
  END; { CASE }

END;  { FUNCTION DOSErrorMsg }
```

As written above, this function converts error codes that are likely to be generated in the program we are developing here. With very little extra effort, you can extend this function to handle all error codes generated by DOS and then use it in other programs that call DOS functions. It will make your program easier to write, easier to debug, and friendlier to use.

RELEASING MEMORY

Due to a peculiarity in DOS, forking a child program from a parent is a two-step process. First, you must release some memory and second, fork the child program. This is because DOS allocates all available memory to the parent, leaving no memory for children.

Turbo Pascal Versions 3.00 and later automatically release excess memory. However, for earlier versions of Turbo, you must deallocate memory yourself. (This feature alone justifies the cost of upgrading to the newer version!)

MAKING FILE CONTROL BLOCKS

If you use the Fork procedure to run programs that operate on files, it may not be enough to pass the filenames via the command line. Some older programs like WordStar expect to find filenames in formatted data areas called File Control Blocks (or FCBs). If you want to fork programs such as children, you have to convert filenames from the command line into FCBs yourself.

DOS provides a function, PARSE FILE NAME (hex 29), to convert a string of characters, as in a command line, into FCBs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here is a routine to convert a command line into two FCBs:

```
TYPE
  FCBType = ARRAY [0..36] OF Char;

PROCEDURE MakeFCB(CmdLine:AnyString; VAR FCB1, FCB2: FCBType);

VAR
  Regs      : RegType;
  DummyFlag : Boolean;

BEGIN { PROCEDURE MakeFCB }

FillChar(FCB1, SizeOf(FCB1), 0);
Regs.AX    := $2901;
Regs.DS    := Seg(CmdLine[1]);
Regs.SI    := Ofs(CmdLine[1]);
Regs.ES    := Seg(FCB1);
Regs.DI    := Ofs(FCB1);
DummyFlag := CallDOS(Regs);

FillChar(FCB2, SizeOf(FCB2), 0);
Regs.AX    := $2901;
Regs.ES    := Seg(FCB2);
Regs.DI    := Ofs(FCB2);
DummyFlag := CallDOS(Regs);

END;  { PROCEDURE MakeFCB }
```

DOS can pass up to two FCBs to a child program. The MakeFCB procedure takes as input a command line and converts the first two tokens it finds into two FCBs. The resulting FCBs are passed back via the VAR parameters FCB1 and FCB2.

Note that after formatting an FCB, DOS advances register SI to point to the next character in the command line. Thus, we do not have to explicitly initialize SI when we format the second FCB.

FORKING A PROGRAM

We now have all the support procedures we need to fork another program. We would like to call the Fork function in the following manner:

```
Flag := Fork('CHKDSK.COM', 'B:');
```

The input to the function should come from two strings, one to specify the name of the child program and another to specify the command line to pass to the child.

The EXEC function in DOS takes its input parameters in a certain way. First, it needs to know the name of the file in which the child program can be found. Second, it requires a parameter block containing the following information:

1. Environment for the child
2. Command line to pass to the child
3. Two FCBs to pass to the child

Every program running under DOS has what is called an environment. This consists of string variables and their values in the form

<variable> = <value>

Some programs have been written so that their behavior varies depending on the value of certain variables in their environment. In our example, we will pass a copy of our own environment to the child, but you can pass an environment of your own design to the child.

To pass a parameter block to the child, we create the following record:

```
VAR
  ParmBlock  : RECORD
                 EnvSeg  : Integer;
                 CmdPtr  : ^AnyString;
                 FCB1Ptr : ^FCBType;
                 FCB2Ptr : ^FCBType;
               END;
```

In the record above, Turbo Pascal pointer fields stand for double word pointers in the Intel segmented address format (offset first, then segment). Here is the Fork function itself:

```
FUNCTION Fork(ProgName, CmdLine : AnyString) : Boolean;

CONST
  CR       = ^M;
  Null     = ^@;


VAR
  Regs : RegType;
  Status : Boolean;
  CmdLen :  Byte ABSOLUTE CmdLine;
  EnvSegAbs : Integer ABSOLUTE CSeg:$002c;
  FCB1, FCB2 :  FCBType;

BEGIN
{ FUNCTION Fork }
  Status := TRUE;

  MakeFCB(CmdLine, FCB1, FCB2);
  ProgName := ProgName + Null;
  CmdLine := ' ' + CmdLine + CR;
  CmdLen := Pred(CmdLen);

  WITH ParmBlock DO
   BEGIN
     EnvSeg := EnvSegAbs;
     CmdPtr := Addr(CmdLine);
     FCB1Ptr := Addr(FCB1);
     FCB2Ptr := Addr(FCB2);
   END;

  Regs.AX := $4b00;

  Regs.BX := Ofs(ParmBlock);
  Regs.DX := Ofs(ProgName[1]);
  Regs.DS := Seg(ProgName[1]);
  Regs.ES := Seg(ParmBlock);

  IF (NOT CallDOS(Regs)) THEN
   BEGIN Status := FALSE;
     WriteLn('Unable to run "', ProgName,
             '" with "', CmdLine, '" : ',
                         DOSErrorMsg(Regs.AX));
   END;
     Fork := Status;
END; { FUNCTION Fork }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Inputs to the Fork function come from two strings: ProgName contains the filename of the program to be forked, while CmdLine contains the command line to be passed to the child program. The Fork function returns true if the child was called successfully, and false otherwise.

You need to set the real constant, TurboVer, to the version number for your copy of Turbo Pascal. If your version is older than 3.00, Fork attempts to release memory before proceeding.

The function converts the program name from a Turbo Pascal string form to the ASCIIZ form required by DOS. This is done by terminating the Pascal string with a null byte.

After two FCBs are parsed off, the command line is converted to a form that DOS requires: It must end in a carriage return, and its first byte is set to the length of the string not counting the terminating carriage return.

The parent program's environment, which is available at location hex 002c of the code segment of every program, is passed to the child via the parameter block. The command line and the two FCBs are also passed to the child via fields in the same parameter block.

You should carefully note how Turbo Pascal strings are passed to DOS functions. The program name itself must be a string of ASCII characters terminated with a null byte (byte of binary zero bits). In DOS jargon, such a string is called an ASCIIZ string. On the other hand, Turbo Pascal strings are stored so that the first byte in storage contains the length of the string, thus obviating the need for a string terminator byte such as a null byte.

To convert a Turbo Pascal string to an ASCIIZ string, append an ASCII null character to the Turbo string. When passing this string to DOS, use the Ofs and Seg built-in procedures on the first character in the string instead of the string name itself.

The command line itself requires a slightly different format than ASCIIZ strings. It should terminate with the ASCII carriage return character, and the first byte of the command line should contain its length not counting the terminating carriage return. Fortunately, this format is the exception rather than the rule.

USING THE FORK FUNCTION

The Fork function gives you a lot of power as a Turbo program designer because now you can build your Turbo program to take advantage of pre-existing programs.

For example, after a particularly disk-intensive operation such as posting journal entries or updating an inventory master file, you can programmatically invoke the CHKDSK program to verify that the directory is clean. For example,

```
Fork('CHKDSK.COM', 'B: /F');
```

For a data communications program, you can programmatically change the baud rate, parity, and stop bit settings by forking the MODE program with a command line:

```
Fork('MODE.COM', 'COM1:1200,N,8,1');
```

If your application requires the user to enter some text that could benefit from formatting, you can invoke a professional word processor from your application for this purpose.

```
Fork('WS.COM', 'SCRATCH.PAD');
```

With little work, you can make your application customizable by letting the user choose his or her favorite word processor. Whenever your application needs such text, it can fork the word processor of the user's choice, passing it the name of a file in its command line. When the user has edited this file, he or she saves it and returns to your application, which can then continue processing. Such customizability has long been the rule in the UNIX and DEC worlds.

WHAT ABOUT DOS INTERNAL COMMANDS?

The Fork function described above can run any program you want, but it cannot execute internal commands such as DIR and TYPE. Such commands are interpreted and executed only by the DOS command interpreter, COMMAND.COM. We now know how to fork programs, so it should be a simple matter of forking COMMAND.COM and asking it to execute such internal commands.

The real question is, where is COMMAND.COM? In the root directory or the current subdirectory? On drive A:, C:, or the currently logged drive? Well, when DOS boots, it automatically defines an environment variable called COMSPEC, whose value contains the drive, path, and filename of the DOS command interpreter. All we have to do is read the value of the COMSPEC environment variable, and we will be able to execute DOS internal commands.

READING THE ENVIRONMENT

We need a function to search the environment for a particular variable such as "COMSPEC" and return its value, as in "A:\COMMAND.COM". A sample call to such a function should look as follows:

```
CommandInterpreter := GetEnv('COMSPEC');
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If the given environment variable is not found, the function returns a null string. Here is the function itself:

```
FUNCTION GetEnv(SearchName : AnyString) : AnyString;

CONST
  Null = ^@;

TYPE
  EnvArrType = ARRAY [0..32767] OF Char;

VAR
  EnvSegAbs  : Integer ABSOLUTE CSeg:$002c;
  EnvArrPtr  : ^EnvArrType;
  Index      : Integer;
  FoundName  : AnyString;
  FoundValue : AnyString;

BEGIN { FUNCTION GetEnv }

EnvArrPtr  := Ptr(EnvSegAbs, $0000);
Index      := 0;
FoundValue := '';

WHILE ((EnvArrPtr^[Index] <> Null) AND (FoundValue = '')) DO
  BEGIN
  FoundName := '';
  WHILE (EnvArrPtr^[Index] <> '=') DO     { copy until = sign }
    BEGIN
    FoundName := FoundName + EnvArrPtr^[Index];
    Index     := Index + 1;
    END;
  IF (FoundName = SearchName) THEN        { found it!         }
    BEGIN
    Index := Index + 1;
    WHILE (EnvArrPtr^[Index] <> Null) DO  { copy until null   }
      BEGIN
      FoundValue := FoundValue + EnvArrPtr^[Index];
      Index      := Index + 1;
      END;
    END
  ELSE                                    { keep looking      }
    BEGIN
    REPEAT
      Index := Index + 1;
      UNTIL (EnvArrPtr^[Index-1] = Null); { advance past null }
    END;
  END;

GetEnv := FoundValue;

END;  { FUNCTION GetEnv }
```

The environment itself contains variables and their values in a particular format, and this function is written to read and understand that format. Each variable and its value are separated by an "=" sign, e.g., PATH=L:\, where PATH is the variable and L:\ is its value. Each variable-value pair is separated from the next variable-value pair by a single null byte. An additional null byte marks the end of the environment.

The function begins by copying the first variable name it finds into a local string called FoundName. If this matches the variable it is looking for, the intervening "=" sign is skipped, and the value is copied into another local string called FoundValue and returned as the value of the function. If it doesn't match, the value is skipped and the next variable name is extracted. The process is repeated until the variable is found or the search is exhausted.

CALLING THE COMMAND INTERPRETER

Given the GetEnv function, invoking the System function itself becomes a simple affair: Fetch the filename where the command interpreter is to be found; modify the command line given by the user so that it begins with the "/C" switch required by the command interpreter; finally, fork the command interpreter with this command line. It looks as follows:

```
FUNCTION System(AnyCommand : AnyString) : Boolean;

VAR
  CommandInt : AnyString;

BEGIN { FUNCTION System }

System     := FALSE;
CommandInt := GetEnv('COMSPEC');

IF (CommandInt = '') THEN
  Writeln('Unable to find command interpreter')
ELSE
  System := Fork(CommandInt, '/c ' + AnyCommand);

END;  { FUNCTION System }
```

The environment variable to be searched for, COMSPEC, must be in all capital letters and must not include the "=" sign that DOS inserts.

The System function will not only execute DOS internal commands, such as DIR and TYPE, but will also give you programmatic access to the power and flexibility of the DOS command interpreter. Facilities such as invoking batch files, searching currently defined paths, redirection, and piping are but a call away.

To print a directory listing without being inundated by the intricacies of DOS function calls, use

```
System('DIR /W');
```

To list a file a screenful at a time, pausing in between for a signal from the user to continue

```
System('MORE < READ.ME');
```

As you have noticed, the System function can run the same programs that the Fork function can. In fact, the System function is much simpler to use — it requires only one argument, and you don't have to specify a .COM or .EXE extension for the program filename. For example, the following two statements are equivalent:

```
Fork('DISKCOPY.COM', 'A: B:');
System('DISKCOPY A: B:');
```

Why, then, use Fork at all? Why not replace it completely by System? In the first place, the Fork function works faster than the System function because it is not encumbered by the need to find, load, and run COMMAND.COM before running the other program. Secondly, programs run by the Fork function have more memory available to them — the command interpreter loaded by the System function consumes about 3K of memory.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A WORD ABOUT USE

Both the Fork and the System functions, together with their support procedures and functions, are given in Listing 1. By putting them all in one file, FORK.INC, you can include them in a program with the following line:

```
{$I FORK.INC}
```

This will keep your main program file small and simple, allowing you to concentrate on the main tasks in your program. At the same time, when needed, the Fork and System functions will be but a call away. This will also allow you to include the Fork and System functions in many programs while maintaining one and only one copy of the source code for these functions.

LIMITATIONS

A common error message you may see from the Fork and System functions is "No more memory." Because the parent program resides in memory, suspended, while the child executes, it is not difficult to run out of memory, particularly for children that require hefty amounts of memory to run. The quickest and easiest workaround for this is to recompile your Turbo program with a lower value of "mAximum free dynamic memory" (in the Options menu when you compile to a .COM file) and try again. In the longer term, it might be advisable to get additional memory for your system.

The Fork and System functions developed here will work only when compiled to a .COM file. Running from memory will have the predictably disastrous effect of crashing your machine. This is because when you run a Turbo Pascal program from memory, the compiler is also resident, making the allocation of memory blocks very different.

Some programs will run as children even with little memory, but are severely limited as a result. Spreadsheet programs like Lotus 123 limit the size of spreadsheets you can create and edit depending on the amount of available memory. Memory-only type editors (the version of EMACS I use) can edit only files which fit in available memory.

CAUTIONS

Because DOS is not the world's best debugged operating system, you may occasionally run into problems like system hangs. When this happens, the best thing to do is to reduce the amount of "maximum free dynamic memory" and recompile your Turbo Pascal program and run it again.

If your system hangs repeatedly, you should also investigate your system's software configuration. I have found that some memory-resident programs like FilePath can cause the system to hang occasionally when memory is scarce.

Various Turbo Pascal hackers around the country have cautioned against using the Turbo MsDos function to perform the DOS EXEC function call because of stack problems on return. I have not had any problems with this version — yet. If your program hangs consistently after the child program terminates, refer to Bela Lubkin's EXEC.PAS program in Data Library 1 of the Turbo Pascal Interest Group on CompuServe. The program contains an example of an InLine procedure to call the DOS EXEC function directly instead of using the Turbo built-in procedure MsDos.

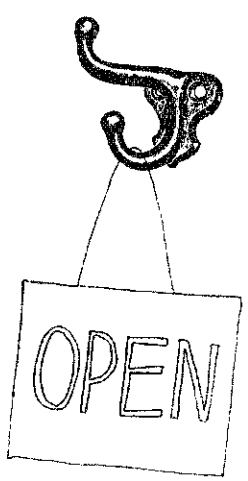
ENHANCEMENTS

A further enhancement to these routines would be to return the error level of the child when it terminates. This value can help your application determine which one of several processing paths to take after forking the child.

CREDITS

I'd like to thank hacker *extraordinaire*, Wayne Chin of Hewlett-Packard Laboratories for his help with research, suggestions, and ideas. These routines were developed on an HP150 Touchscreen Max computer using Turbo Pascal Version 3.01A. They were tested on an IBM PC/XT using Turbo Pascal Versions 3.00B and 3.01A, and on the HP150 using Turbo Pascal Version 2.00B. □

Comments and questions can be sent to Arvind at 3500 Deer Creek Road, Palo Alto, CA 94304, or via Usenet at .../ucbvax!hplabs!kumar.



NORMAL OPEN STATEMENT

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

③

```
/*
|-------------------------------------------------------------
|  sort by insertion
|   make root and first node of singly-linked list into root pointer
|   and root of tree, respectively
|-------------------------------------------------------------
*/
        list = (root->llink)->llink;
        /* set list pointer to 2nd node of list;*/
           /* 2nd node becomes head of list */
                /* set link at 1st node to NULL; 1st node becomes tree root */
        (root->llink)->llink = (struct tnode *)NULL;

        while(list != (struct tnode *)NULL) {
        /* continue until last node in list found */

                next = list->llink;
                /* next points to next node in list */
                parent = root;
                /* parent points to parent of rebal */
                rebal = root->llink;
                /* rebal points to node where rebalancing may be needed */

                current = root->llink;
                /* current points to the current node */
                w = list->point;
                /* get next word */

                search(current,rebal,parent,w,list);
                /* find the location, recursive */
                list = next;
                /* set list to point to next node in list */
        }

/*
|-------------------------------------------------------------
|  write sorted list
|-------------------------------------------------------------
*/
        treeprint(root->llink);
/*
|-------------------------------------------------------------
|  shutdown, close files, print messages
|-------------------------------------------------------------

*/
        shutdown();

}

/*
|-------------------------------------------------------------
|  function definitions
|-------------------------------------------------------------
*/

shutdown()
{
        fclose(infp);
        fclose(outfp);
}


char *getword(s)   /* gets a word from the input file */
char    *s;                /* s points to array to receive word */
{                                  /* returns pointer to array or NULL */
        some function for reading a word and returning a pointer to it
}

/*
|-------------------------------------------------------------
|  search() finds the location for w at or below current
|   this implementation of search is recursive
|-------------------------------------------------------------
*/
search(current,rebal,parent,w,list)
struct  tnode   *current,*rebal,*parent,*list;
char    *w;
{
        int             i;
        struct  tnode   *new;
```

④

```
        i = strcmp(w,current->point);

        if (i < 0) {                            /* go left */
                new = current->llink;
                if (new == (struct tnode *)NULL) {
                /* current is a terminal node */
                        new = list;
                        /* insert node from list */
                        current->llink = new;
                        new->llink = (struct tnode *)NULL;
                        /* A5; mark as terminal node */
                        new->rlink = (struct tnode *)NULL;
                        new->b = 0;
                        /* set balance factor to zero */
                        rot_bal(new,current,rebal,parent,w);
                        /* Rotate and balance */


                }
                else {
                /* current is not a terminal node */
                        if (new->b != 0) {
                                parent = current;
                                rebal = new;
                        }
                        current = new;
                        search(current,rebal,parent,w,list);
                }
        }
        else if (i > 0) {                       /* go right */
                new = current->rlink;
                if (new == (struct tnode *)NULL) {
                /* current is a terminal node */
                        new = list;
                        /* insert node from list */
                        current->rlink = new;
                        new->llink = (struct tnode *)NULL;
                        /* A5; mark as terminal node */
                        new->rlink = (struct tnode *)NULL;
                        new->b = 0;
                        /* set balance factor to zero */
                        rot_bal(new,current,rebal,parent,w);
                        /* rotate and balance */
                }
                else {
                /* current is not a terminal node */
                        if (new->b != 0) {
                                parent = current;
                                rebal = new;
                        }
                        current = new;
                        search(current,rebal,parent,w,list);
                }
        }
        else {
        /* if (strcmp(w,current->point) == 0)  word is already in tree */
                return structures at:
                        list
                        list->point
                to the free memory pool
        }
        return;
}

/*
|-------------------------------------------------------------
|  rot_bal adjusts the balance factors from rebal to the
|   newly inserted node, and then rotates the tree if necessary
|   (follows Knuth, Algorithm A, pp. 455ff., Vol. 3)
|-------------------------------------------------------------
*/

rot_bal(new,current,rebal,parent,w)
struct  tnode   *new,*current,*rebal,*parent;
char            *w;
{
        struct  tnode   *rebalnode;
        /* rebalnode marks the node where rebalancing begins */

        int             a;
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
if (strcmp(w,rebal->point) < 0) {
/* A6: adjust the balance factors */
        rebalnode = rebal->llink;
        current = rebal->llink;
}
else {
        rebalnode = rebal->rlink;
        current = rebal->rlink;
}
while (current != new) {
        if (strcmp(w,current->point) < 0) {
                current->b = -1;
                /* current's subtree heavy on left */
                current = current->llink;
        }
        else {  /* if (strcmp(w,current->point) > 0) */
                current->b = 1;
                /* current's subtree heavy on right */
                current = current->rlink;
        }
}

if (strcmp(w,rebal->point) < 0)
/* A7: final balance and decision on rotation */

        a = -1;
else
        a = 1;

if (rebal->b == 0) {
        rebal->b = a;
        root->b++;      /* tree is taller; increment height */
        return;         /* and quit */
}
else if (rebal->b == -a) {
        rebal->b = 0;   /* tree is more balanced */
        return;                 /* quit */
}
else if (rebal->b == a) {       /* tree is out of balance */
        if (rebalnode->b == a) {
        /* A8: make a single rotation */
                current = rebalnode;
                if (a == -1) {
                        rebal->llink = rebalnode->rlink;

                        rebalnode->rlink = rebal;
                }
                else {
                        rebal->rlink = rebalnode->llink;
                        rebalnode->llink = rebal;
                }
                rebal->b = 0;
                rebalnode->b = 0;
        }
        else if (rebalnode->b == -a) {
        /* A9: make a double rotation */
                if (a == -1) {
                        current = rebalnode->rlink;
                        rebalnode->rlink = current->llink;
                        current->llink = rebalnode;
                        rebal->llink = current->rlink;
                        current->rlink = rebal;
                }
                else {
                        current = rebalnode->llink;
                        rebalnode->llink = current->rlink;
                        current->rlink = rebalnode;
                        rebal->rlink = current->llink;
                        current->llink = rebal;
                }
                if (current->b == a) {
                        rebal->b = -a;
                        rebalnode->b = 0;
                }
                else if (current->b == 0) {
                        rebal->b = 0;
                        rebalnode->b = 0;
                }
                else {  /* if (current->b == -a) */
                        rebal->b = 0;
                        rebalnode->b = a;
                }
                current->b = 0;
        }
        /* A10: connect parent to new child after rebalancing */
        if (rebal == parent->rlink)

                parent->rlink = current;
        else
                parent->llink = current;
    }
}

struct  tnode   *talloc()       /* K & R, p. 134 */
{
        return((struct tnode *)allocation function(sizeof(struct tnode)));
}
```

```
/*
|-------------------------------------------------------------------
|  print tree at current recursively; K & R, p. 133, with mods
|-------------------------------------------------------------------
*/
treeprint(current)
struct  tnode   *current;
{
        if (current != (struct tnode *)NULL) {
                treeprint(current->llink);
                print the word at current->point
                treeprint(current->rlink);
        }
}

/*
|-------------------------------------------------------------------
|  strcmp() is a string compare function
|   compares in strictly ASCII order; register variables for speed
|-------------------------------------------------------------------
*/
strcmp(s1,s2)
register char *s1,*s2;
{
        while((*s1 == *s2) && *s1) {
                s1++;
                s2++;
        }
        return(*s1 - *s2);
}

quit(reason)
char    *reason;
{
        int     code = 1;       /* dummy error code */

        printf("%s\n\n",reason);
        shutdown();
        _exit(code);
}
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

①

```
{ File FORK.INC }

CONST
  CR       = ^M;
  Null     = ^@;


TYPE
  AnyString = String[255];

VAR
  EnvSegAbs  : Integer ABSOLUTE CSeg:$002c;   { environment str seg }


FUNCTION DOSErrorMsg(AX : Integer) : AnyString;

VAR
  ErrNumStr : AnyString;

BEGIN { FUNCTION DOSErrorMsg }

CASE AX OF
  1  : DOSErrorMsg := 'Invalid function';
  2  : DOSErrorMsg := 'File not found';
  3  : DOSErrorMsg := 'Path not found';
  4  : DOSErrorMsg := 'Too many open files';
  5  : DOSErrorMsg := 'Access denied';
  7  : DOSErrorMsg := 'Arena trashed';
  8  : DOSErrorMsg := 'Not enough memory';
  9  : DOSErrorMsg := 'Invalid block';
  10 : DOSErrorMsg := 'Bad environment';
  11 : DOSErrorMsg := 'Bad format';
  ELSE
   BEGIN
   Str(AX, ErrNumStr);
   DOSErrorMsg := 'DOS Error ' + ErrNumStr;
   END;
  END; { CASE }

END;  { FUNCTION DOSErrorMsg }


FUNCTION Fork(ProgName, CmdLine : AnyString) : Boolean;

TYPE
  RegType = RECORD
              AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer;
            END;
  FCBType = ARRAY [0..36] OF Char;

VAR
  Regs       : RegType;
  Status     : Boolean;
  CmdLen     : Byte ABSOLUTE CmdLine;
  FCB1, FCB2 : FCBType;
  ParmBlock  : RECORD
                 EnvSeg  : Integer;
                 CmdPtr  : ^AnyString;
                 FCB1Ptr : ^FCBType;
                 FCB2Ptr : ^FCBType;
               END;


  FUNCTION CallDOS(VAR Regs : RegType) : Boolean;

  BEGIN { FUNCTION CallDOS }
```

②

```
  MsDos(Regs);
  CallDOS := (NOT Odd(Regs.Flags));

  END;  { FUNCTION CallDOS }


  PROCEDURE MakeFCB(CmdLine : AnyString; VAR FCB1, FCB2 : FCBType);

  VAR
    Regs      : RegType;
    DummyFlag : Boolean;

  BEGIN { PROCEDURE MakeFCB }

   FillChar(FCB1, SizeOf(FCB1), 0);
   Regs.AX   := $2901;
   Regs.DS   := Seg(CmdLine[1]);
   Regs.SI   := Ofs(CmdLine[1]);
   Regs.ES   := Seg(FCB1);
   Regs.DI   := Ofs(FCB1);
   DummyFlag := CallDOS(Regs);

   FillChar(FCB2, SizeOf(FCB2), 0);
   Regs.AX   := $2901;
   Regs.ES   := Seg(FCB2);
   Regs.DI   := Ofs(FCB2);
   DummyFlag := CallDOS(Regs);

   END;  { PROCEDURE MakeFCB }


BEGIN { FUNCTION Fork }

Status := TRUE;

  MakeFCB(CmdLine, FCB1, FCB2);
  ProgName  := ProgName + Null;
  CmdLine   := ' ' + CmdLine + CR;
  CmdLen    := Pred(CmdLen);
  WITH ParmBlock DO
    BEGIN
    EnvSeg  := EnvSegAbs;
    CmdPtr  := Addr(CmdLine);
    FCB1Ptr := Addr(FCB1);
    FCB2Ptr := Addr(FCB2);
    END;

  Regs.AX := $4b00;
  Regs.BX := Ofs(ParmBlock);
  Regs.DX := Ofs(ProgName[1]);
  Regs.DS := Seg(ProgName[1]);
  Regs.ES := Seg(ParmBlock);

  IF (NOT CallDOS(Regs)) THEN
    BEGIN
    Status := FALSE;
    WriteLn('Unable to run "', ProgName, '" with "', CmdLine, '" : ',
          DOSErrorMsg(Regs.AX));
    END;

  END; { IF }

Fork := Status;

END;  { FUNCTION Fork }
```

③

```
FUNCTION System(AnyCommand : AnyString) : Boolean;

  VAR
    CommandInt : AnyString;


  FUNCTION GetEnv(SearchName : AnyString) : AnyString;

  TYPE
    EnvArrType = ARRAY [0..32767] OF Char;

  VAR
    EnvArrPtr   : ^EnvArrType;
    Index       : Integer;
    FoundName   : AnyString;
    FoundValue  : AnyString;

  BEGIN { FUNCTION GetEnv }

  EnvArrPtr   := Ptr(EnvSegAbs, $0000);
  Index       := 0;
  FoundValue  := '';

  WHILE ((EnvArrPtr^[Index] <> Null) AND (FoundValue = '')) DO
    BEGIN
    FoundName := '';
    WHILE (EnvArrPtr^[Index] <> '=') DO          { copy until = sign }
      BEGIN
      FoundName := FoundName + EnvArrPtr^[Index];
      Index     := Index + 1;
      END;
    IF (FoundName = SearchName) THEN
      BEGIN
      Index := Index + 1;                        { advance past = sign}
      WHILE (EnvArrPtr^[Index] <> Null) DO       { copy until null    }
        BEGIN
        FoundValue := FoundValue + EnvArrPtr^[Index];
        Index      := Index + 1;
        END;
      END
    ELSE
      BEGIN
      REPEAT
        Index := Index + 1;                      { advance PAST null }
        UNTIL (EnvArrPtr^[Index-1] = Null);
      END;
    END;

  GetEnv := FoundValue;

  END;  {FUNCTION GetEnv }


BEGIN { FUNCTION System }

System      := FALSE;
CommandInt := GetEnv('COMSPEC');

IF (CommandInt = '') THEN
  Writeln('Unable to find command interpreter.')
ELSE
  System := Fork(CommandInt, '/c ' + AnyCommand);

END;  { FUNCTION System }
```


Yes!

Please enter my new subscription to

programmer's journal

Bi-Monthly Subscription Rate

	U.S.	Canada (U.S. Funds only)	Foreign
1 year	☐ $24	☐ $30	☐ $40
2 years	☐ $44	☐ $55	☐ $75
3 years	☐ $64	☐ $80	☐ $110

☐ New Subscription
☐ Renewal

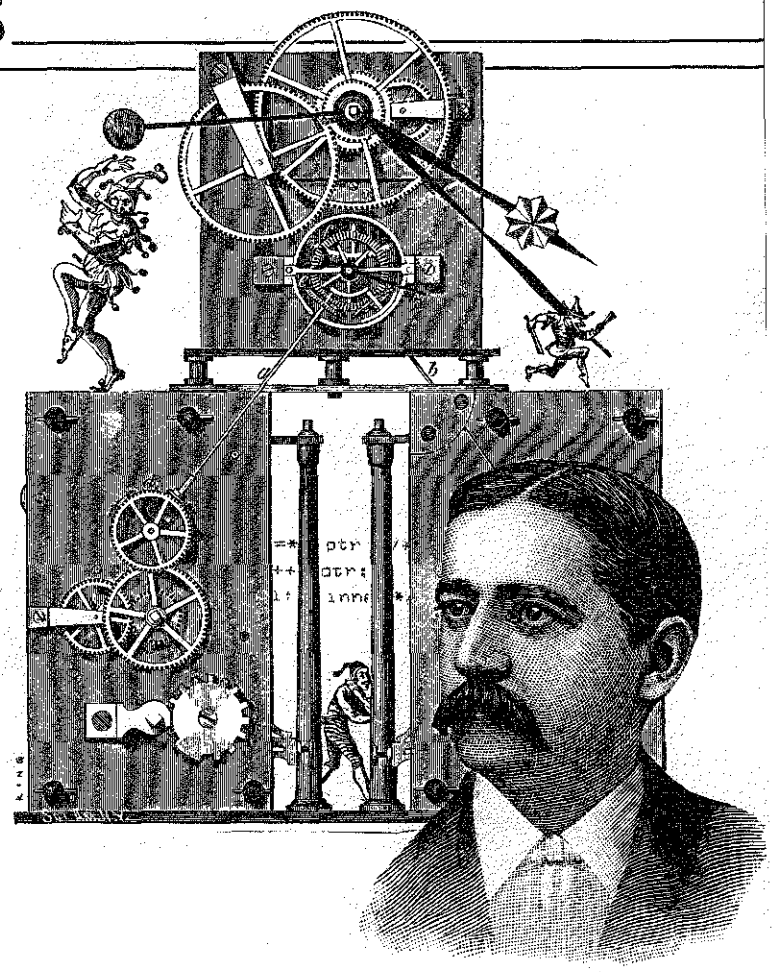
☐ Check enclosed
☐ Bill me
☐ Charge my Bankcard: ☐ VISA ☐ MC
Card No. _____
Expiration date _____

Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Telephone _____

Mail to: P.O. Box 3000 Department EE
Denville, NJ 07834

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



AN OPERATING SYSTEM FOR THE PEOPLE

by Hal Hardenbergh

The company of which I am president owns three Atari 520 STs. The operating system and (preliminary) BASIC eat up almost all of the 512K memory involved! Why?

I am concerned that the personal computer industry seems to be losing sight of its origins. Who remembers that the beginnings of the personal computer industry featured very efficient use of its CPU and memory resources and most of all independence of the computer professional in his symbolic white coat? Today, things are virtually reversed.

Nobody in the personal computer industry cares about efficient use of CPU and memory resources any more. The original heroes of the personal computer revolution — Jack Tramiel, DRI, Apple — have donned white coats and an attitude of superiority and have become the enemy.

The Atari 520 ST has integer computing capacity about equal to a VAX ll/750 or half a VAX ll/780. So with the pre-release Atari BASIC a program such as:

```
10 PRINT "THIS IS BASIC"
20 GOTO 10
```

executes so slowly that the lines can easily be counted out loud as they appear! One iteration of *BYTE*'s January '83 Sieve — that's ONE iteration — requires 97 seconds to run! Why? Because the computer professionals in their white coats have decided that it is all right — even preferred — to write operating systems and languages for the 68000 *in a high level language!* That is why Atari's ST BASIC runs so slowly, and why the operating systems and BASIC eat up almost all of the 512K RAM.

Not only that, but these same professionals have apparently decided that personal computer users (68000 division) are all illiterates and that pictograms and icons are all that they can comprehend! After all, if they had any intelligence, they would learn Modula-2 and get their very own white coats, right?

IBM'S OPERATING SYSTEM

Has anybody figured out that if you hand a $150,000 per year orthopedic surgeon (or investment broker, or performing artist) an operating system with a 3,000 page manual that it will require nearly full-time use of that operating system to even REMEMBER the commands and syntax from day to day? Do you realize that an orthopedic surgeon who uses a computer full-time has become a $5/hour key-to-disk operator?

And yet Microsoft's Bill Gates is hell-bent on taking the most successful operating system of all time (above the C-64 level, that is) and TURNING IT INTO UNIX. WHY? I guess, because if the personal computer owner had any intelligence or self-respect, he would learn and use Modula-2 *and* UNIX!

DISCLOSURE STATEMENT: Digital Acoustics is developing a product along the lines which I am going to recommend. However, the product is 68000-based and hence not directed toward the main readership of the *Programmer's Journal* — so please forgive what under other circumstances could be considered an unpaid self-advertisement.

In fact, I'm knowingly presenting this opinion to a largely hostile audience. The readers of *Programmer's Journal* are likely to include a large proportion of persons wearing psychic white vestments that were acquired over the past several years of full-time programming for the IBM PC.

Therefore, I hoist an industrial-grade bumbershoot to ward off rotten vegetables and other rude objects that might precipitate from my audience and assert: *The operating systems which are popular amongst the professional programming community for the personal computer field are*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ATTENTION
C-PROGRAMMERS
File System Utility Libraries
All products are written entirely in K&R C. Source code included, No Royalties, Powerful & Portable.
BTree Library
75.00
• High speed random and sequential access.
• Multiple keys per data file with up to 16 million records per file.
• Duplicate keys, variable length data records.
ISAM Driver
40.00
• Greatly speeds application development.
• Combines ease of use of database manager with flexibility of programming language.
• Supports multi key files and dynamic index definition.
• Very easy to use.
Make
59.00
• Patterned after the UNIX utility.
• Works for programs written in every language.
• Full macros, File name expansion and built in rules.
Full Documentation and Example Programs Included.
ALL THREE PRODUCTS FOR —
149.00
For more information call or write:
1343 Stanbury Drive
Oakville, Ontario, Canada
L6L 2J5
(416) 825-0903
softocus
Credit cards accepted.
Dealer inquiries invited.
Circle **22** on the reader service card

much too complicated for the average user of these systems. (In addition, the 68000 world is cursed by white-coated professionals who just KNOW that the chip is so fast that everything can be done in a high-level language, including operating systems and an interactive BASIC!) Let's let the 68000 go for now and concentrate on operating systems for the IBM PC and its relations.

Because you readers are mostly full-time computer professionals, the best operating system for you is UNIX. Honest! I have been told many times over the past four years that UNIX is wonderful and that the 3000-page manual is even more wonderful, and I have no doubt that this is all true — for full-time computer professionals.

But to ask a $150,000/year orthopedic surgeon to devote the time necessary to learn that 3000 page UNIX manual is ridiculous. Even if he took a two-week vacation and learned it — orthopedic surgeons are not stupid — he would never operate the system enough hours on an on-going basis to retain all the cryptic commands UNIX features.

While it is not true that the average IBM PC user earns $150,000 per year, it IS true that that user operates his personal computer for an hour or less a day. (Ducking a fresh flight of turnips, I hasten to assure my audience that I am NOT including you professional programmers in the category of "average user.") What the average personal computer user needs is not UNIX, or even a version of MS-DOS approaching UNIX, but *an operating system appropriate for an intelligent, thoughtful user who uses his personal computer for an hour a day.*

Yes, there are a limited few small computers out there being used as *business computers*, and those get used a lot more than an hour a day. But there are ten to a hundred *personal* computers for each business computer! The operating systems for the large mass of personal computers should not be overladen with excessive complexity for the convenience of the limited few business computer users.

AN OPERATING SYSTEM FOR USERS

Now I will outline the details of an operating system suitable for the average personal computer user. The functions needed in this system are

```
LOAD filename[,drive#]
SAVE filename[,drive #]
RUN filename[,drive #]

DATASAVE filename[(drive #)],parameter list
DATALOAD filename[(drive #)],parameter list

BLOAD filename[(drive #)][,address]
BSAVE filename[(drive #)],address,length
BRUN filename[(drive #)][,address]

SECTORLOAD sector #,stringarray
SECTORSAVE sector #,stringarray

[UN]LOCK filename[,drive #]
[UN]SCRATCH filename[,drive #]

DIRectory, DIR1, DIR2...

BACKUP[src drive, dest drive]
COPY[src drive, dest drive]
```

BACKUP copies a mirror image of a floppy disk from one drive to another. COPY copies files one by one, files that have not been SCRATCHED. To SCRATCH a file is to mark it for deletion upon the occasion of the next COPY. LOCK renders a file "read only." Naturally, some of the above operating system functions will need to be program statements (DATALOAD/SAVE) or to be replicated as program statements in the interactive BASIC that should accompany this operating system. SECTORLOAD/SAVE allows all sorts of disk/disk catalog manipulations/fixes. Naturally, the stringarray should be dimensioned to the same size as a sector on the host computer.

Those of you who believe that Moses came down from Ararat

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

with MS-DOS engraved on the back of those two stone tablets will be puzzled by the syntax of DATALOAD/SAVE. Here is the CORRECT way to save (for example) a floating point varn, an integer array, and a string array on Disk 2:

DATASAVE filename (D2),fp,int%(),string$()

The lads from Bellevue have been screwing up this simple operating system function for a decade now! (PRINTing to disk and INPUTing from disk indeed!)

HARD DISKS? Glad you asked. If we have a hard disk, we will most certainly need multiple directories and a simple — I emphasize SIMPLE — method of switching from one directory to another. However, in late 1985/early 1986 I question just how many *average personal computer users* will have a hard disk. (Yes, I know most of you in my audience have or want a hard disk, but you are NOT average!) Small computers with hard disks — and enough data to actually require hard disks — tend to be business computers that are used many hours per day, hence, outside the purview of this argument.

Because our hypothetical orthopedic surgeon is liable to go three or four weeks occasionally without using his personal computer, we should provide a plastic-sheathed "cheat sheet" with the operating system commands and example syntax to be left alongside the personal computer for quick reference.

Now if all of you out there will suppress your howls of protest — yes, I know about things like command line interpreters and batch files, and I know about track 0 being reserved for a command line in ancient CP/M. I have even heard of multi-tasking, which is ridiculous for a personal computer that is powered down 23 hours out of every 24. As a professional programmer complete with real or psychic white coat YOU surely feel the occasional need of such! But — DOES YOUR CUSTOMER? And are we providing a DOS for YOU or for your CUSTOMER?

To answer that question is to judge the wisdom of turning MS-DOS into a progressively more complex operating system whose goal is to become UNIX. □

Highly skeptical primate ponders the cryptic meaning of 'Volume in Drive A: has no label.'

NEW!

Advanced Trace86™

Symbolic Debugger & Assembler Combo

- Full-screen trace with single stepping; Even backstepping!
- Write & Edit COM & EXE programs
- Conditional breakpoints (programmable)
- Switch between trace and output screen; Or set up two monitors
- 8087, 80186, 80286, 80287 support
- Write labels & comments on code
- Polish hex/decimal calculator
- and more . . . Priced at $175.00

To order or request more information contact:

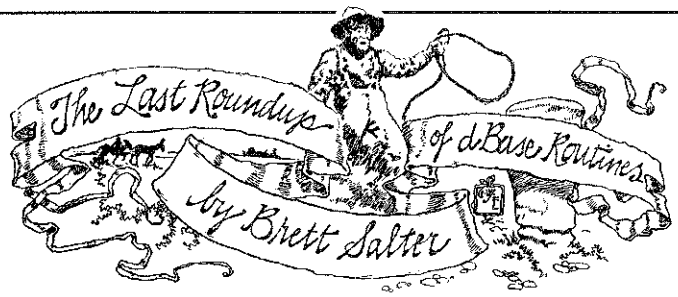
Morgan Computing Co., Inc.

P.O. Box 112730, Carrollton, TX 75011

(214) 245-4763

Circle **14** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



THE FINAL INSTALLMENT OF BRETT SALTER'S dTOOLS

Here are the dBASE II calling programs for the two previous issues' assembly language routines, except for ANSI and LP. The assembly code is held as data bytes in the dBASE II program and is poked into memory before use. (The DBFIX listing in Volume 3, Number 5 has no utilizing dBase program — it is compiled and invoked from DOS.)

Brett Salter points out that we let the hard tabs slip out of the assembly listings in the last two issues. For the corrected assembly listings and associated dBase routines on disk, send $8 to *PJ*.

The disk contains the additional routines KEYFIX, LP2, EXEC, SOUND, MSTRING, PEEK, POKE, and CONFIG. Brett tells us that he had originally planned to market the whole package, but became involved in other things. □

Copyright 1983 - Data Base Decisions, Atlanta, GA

```
* COMPRESS.PRG - compress a byte
*
* Load the subroutine
* Set poke address (must be at least 57,344)
set talk off
set bell off
store 57344 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 1, 0, 232, 0, 0, 94, 131, 238, 4
poke subloc+ 10, 176, 0, 136, 4, 138, 15, 128, 249, 8, 119
poke subloc+ 20, 29, 181, 0, 67, 176, 0, 138, 39, 128, 252
poke subloc+ 30, 48, 114, 17, 128, 252, 49, 119, 12, 128, 236
poke subloc+ 40, 48, 208, 224, 2, 196, 67, 226, 234, 136, 4
poke subloc+ 50, 195
*
* Run the subroutine
* Establish argument(s)
store y to continue
do while continue
erase
store '        ' to string
@ 1,1 say 'COMPRESS - compress a bit string into a byte'
@ 4,1 say 'Enter bit string of zeroes and ones ' get string
read
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call string
*
* Display returned value
@ 6,1 say 'The byte is:'
@ 6,13 say peek(subloc+2)
@ 9,1 say 'Continue (y/n)? ' get continue
read
enddo
return
```

```
* EXPAND.PRG - expand a byte
*
* Load the subroutine
* Set poke address (must be at least 57,344)
set talk off
set bell off
store 57344 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 1, 0, 232, 0, 0, 94, 131, 238, 4
poke subloc+ 10, 138, 7, 60, 8, 117, 27, 138, 4, 67, 185
poke subloc+ 20, 8, 0, 180, 128, 186, 49, 48, 132, 196, 116
poke subloc+ 30, 5, 136, 23, 235, 3, 144, 136, 55, 67, 208
poke subloc+ 40, 236, 226, 240, 195
*
* Run the subroutine
* Establish argument(s)
store y to continue
do while continue
erase
store 0 to byte
store '........' to answer
@ 1,1 say 'EXPAND - expand a byte'
@ 4,1 say 'Enter number (0 to 255) ' get byte picture '999'
read
*
* Compute and poke segment and offset
poke subloc+2,byte
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call answer
*
* Display returned value
@ 6,1 say 'The bits are:'
@ 6,14 say answer
@ 9,1 say 'Continue (y/n)? ' get continue
read
enddo
return
```

```
* LPTSWAP.PRG - swap parallel printers 1 and 2
*
* This program is used to logically swap parallel printers
* 1 and 2. The first time the program is run, lpt2 becomes
* lpt1 and vice-versa. The second time the program is run,
* the printers are swapped back to their original state.
*
* Load the subroutine
* Set poke address (must be at least 57,344)
set talk off
set bell off
store 57344 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 6, 51, 192, 142, 192, 191, 8, 4, 38, 139
poke subloc+ 10, 5, 38, 139, 93, 2, 38, 137, 29, 38, 137
poke subloc+ 20, 69, 2, 7, 195
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
return
```

```
* IN.PRG - retrieve the value of a port
*
* Load the subroutine
* Set poke address (must be at least 56,832)
set talk off
store 56832 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 3, 0, 0, 0, 232, 0, 0, 94, 131
poke subloc+ 10, 238, 6, 139, 20, 236, 136, 68, 2, 195
*
* Run the subroutine
* Establish argument(s)
store 0 to port
erase
@ 1,20 say 'Read a machine port'
@ 6,20 say 'Enter port number' get port picture '99999'
read
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
*
* Compute and poke port number
store int(port/256) to porthi
store port-256*porthi to portlo
*
poke subloc+2,portlo
poke subloc+3,porthi
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
* Display returned value
@ 8,20 say 'The port contains'
@ 8,38 say peek(subloc+4) using '999'
wait to return
return


* OUT.PRG - change the value of a port
*
* Load the subroutine
* Set poke address (must be at least 56,832)
set talk off
store 56832 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 3, 0, 0, 0, 232, 0, 0, 94, 131
poke subloc+ 10, 238, 6, 139, 20, 138, 68, 2, 238, 195
*
* Run the subroutine
* Establish argument(s)
store 0 to port,value
erase
@ 1,20 say 'Write a machine port'
@ 6,20 say 'Enter port number' get port picture '99999'
@ 8,20 say 'Enter value      ' get value picture '999'
read
*
* Compute and poke port number and value
store int(port/256) to porthi
store port-256*porthi to portlo
*
poke subloc+2,portlo
poke subloc+3,porthi
poke subloc+4,value
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
* Display returned value
@ 10,20 say 'Port changed'
wait to return
return


* READNO.PRG - read a number from a memvar
*
* Load the subroutine
* Set poke address (must be at least 57,344)
set talk off
set bell off
store 57344 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 6, 0, 0, 78, 85, 77, 0, 232, 0
poke subloc+ 10, 0, 95, 131, 239, 9, 185, 64, 0, 190, 0
poke subloc+ 20, 154, 139, 4, 59, 69, 2, 117, 8, 139, 68
poke subloc+ 30, 2, 59, 69, 4, 116, 8, 131, 198, 16, 226
poke subloc+ 40, 236, 235, 59, 144, 138, 68, 11, 60, 206, 117
poke subloc+ 50, 51, 139, 116, 13, 138, 4, 44, 63, 114, 42
poke subloc+ 60, 138, 232, 51, 192, 183, 0, 70, 138, 28, 177
poke subloc+ 70, 4, 210, 235, 232, 31, 0, 254, 205, 116, 17
poke subloc+ 80, 138,  28, 177, 4, 210, 227, 210, 235, 232, 16
poke subloc+ 90, 0, 254, 205, 116, 2, 235, 225, 137, 5, 235
poke subloc+100, 5, 144, 51, 192, 137, 5, 195, 81, 185, 10
poke subloc+110, 0, 247, 225, 89, 3, 195, 195
*
* Run the subroutine
* Establish argument(s)
store y to continue
do while continue
store 0 to num
erase
@ 1,1 say 'Read a numeric memvar'
@ 6,1 say 'Enter number' get num picture '99999.99'
read
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
* Display returned value
@ 8,1 say 'The number is'
@ 8,15 say peek(subloc+2)+256*peek(subloc+3) using '99999'
@ 10,1 say 'Continue (y/n)?' get continue
read
enddo

wait to return
return


* TIME.PRG - get the system time
*
* Load the subroutine
* Set poke address (must be at least 56,832)
set talk off
store 56832 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 138, 7, 60, 8, 117, 36, 180, 44, 205, 33
poke subloc+ 10, 138, 197, 232, 28, 0, 137, 71, 1, 181, 58
poke subloc+ 20, 136, 111, 3, 138, 193, 232, 15, 0, 137, 71
poke subloc+ 30, 4, 136, 111, 6, 138, 198, 232, 4, 0, 137
poke subloc+ 40, 71, 7, 195, 212, 10, 5, 48, 48, 134, 224
poke subloc+ 50, 195
*
* Run the subroutine
* Establish argument(s)
store '........' to time
erase
@ 1,20 say 'Read the time from DOS'
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call time
*
* Display returned value
@ 6,20 say 'The time is'
@ 6,32 say time
wait to return
return


* DOSBIOS.PRG - perform DOS & BIOS function calls
*
* Load the subroutine
* Set poke address (must be at least 56,832)
set talk off
store 56832 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 9, 0, 0, 0, 0, 0, 0, 0, 0
poke subloc+ 10, 0, 232, 0, 0, 94, 131, 238, 12, 138, 68
poke subloc+ 20, 8, 232, 0, 0, 91, 129, 195, 20, 0, 136
poke subloc+ 30, 7, 139, 4, 139, 92, 2, 139, 76, 4, 139
poke subloc+ 40, 84, 6, 86, 205, 0, 94, 137, 4, 137, 92
poke subloc+ 50, 2, 137, 76, 4, 137, 84, 6, 195
*
* Run the subroutine
* Establish argument(s)
erase
@ 1,20 say 'DOS/BIOS function calls'
poke subloc+2,0,6,0,112,0,0,79,24,16
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
* Display returned values
*@  6,20 say 'Number of disk drives     - '
*@  8,20 say 'Video type                - '
*@ 10,20 say 'System board memory       - '
*@ 12,20 say 'Parallel printer adapters - '
*@ 14,20 say 'Game control adapters     - '
*@ 16,20 say 'Communications adapters   - '
wait to return
return
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Write it once!

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the art implementation of the Forth computer language. Forth-is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC family (MS DOS 2.1)	125
Apple II family (DOS 3.3)	125
CP/M 2.x (IBM 3740 8")	125
Commodore 64 (with graphics)	100
Extensions	
Software Floating Point	$60
Hardware (8087) Floating Point	60
Graphics (selected systems)	60
Relocator (with utility sources)	60
Target compiler (RAM and ROM)	350
Target Application Generation System (TAGS) - MasterFORTH, Target compiler & relocator	495
Publications	
Printed source listings (each)	$15
Forth-83 International Standard	*18*

NEW PRODUCTS

MODEL List Processor $40
(with ELIZA and micro-LISP)

MICROMOTION

8726 S. Sepulveda Blvd. #A171
Los Angeles, CA 90045
(213) 827-4340

Circle **13** on the reader service card


```
* ANDOR.PRG - manipulate bits within a byte
*
* Load the subroutine
* Set poke address (must be at least 57,344)
set talk off
set bell off
store 57344 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 235, 4, 0, 0, 0, 0, 232, 0, 0, 94
poke subloc+ 10, 131, 238, 7, 139, 68, 1, 138, 28, 128, 251
poke subloc+ 20, 97, 114, 3, 128, 235, 32, 128, 251, 65, 116
poke subloc+ 30, 17, 128, 251, 79, 116, 17, 128, 251, 88, 116
poke subloc+ 40, 17, 179, 63, 136, 4, 235, 19, 144, 34, 224
poke subloc+ 50, 235, 11, 144, 10, 224, 235, 6, 144, 50, 224
poke subloc+ 60, 235, 1, 144, 136, 100, 3, 195
*
* Run the subroutine
* Establish argument(s)
store y to continue
do while continue
erase
store ' ' to function
store 0 to byte1, byte2, result
@ 1,1 say 'ANDOR - manipulate bits within a byte'
@ 4,1 say 'Enter function (A=and, O=or, X=xor) ' get function
@ 5,1 say 'Enter first byte (0 to 255)          ' get byte1 picture '999'
@ 6,1 say 'Enter second byte (0 to 255)         ' get byte2 picture '999'
read
*
* Poke arguments
poke subloc+2,rank(function)
poke subloc+3,byte1,byte2
*
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call
*
* Display result
@ 8,1 say 'The result is:'
@ 8,16 say peek(subloc+5)
@ 10,1 say 'Continue (y/n)? ' get continue
read
enddo
return



* KEYBOARD.PRG - read/change the keyboard
*
* Load the subroutine
* Set poke address (must be at least 56,832)
set talk off
store 56832 to subloc
*
* Poke subroutine into memory, starting at subloc
poke subloc+  0, 138, 7, 60, 9, 117, 88, 6, 51, 192, 142
poke subloc+ 10, 192, 191, 23, 4, 38, 138, 37, 7, 138, 71
poke subloc+ 20, 1, 185, 8, 0, 139, 243, 70, 70, 60, 82
poke subloc+ 30, 116, 7, 60, 67, 116, 28, 235, 56, 144, 176
poke subloc+ 40, 128, 182, 48, 178, 49, 132, 224, 117, 5, 136
poke subloc+ 50, 52, 235, 3, 144, 136, 20, 70, 208, 232, 226
poke subloc+ 60, 240, 235, 31, 144, 180, 0, 176, 128, 138, 20
poke subloc+ 70, 128, 250, 49, 117, 2, 10, 224, 70, 208, 232
poke subloc+ 80, 226, 242, 6, 51, 219, 142, 195, 191, 23, 4
poke subloc+ 90, 38, 136, 37, 7, 195
*
* Run the subroutine
* Establish argument(s)
store ' ' to function
erase
@ 1,20 say 'Read/change the keyboard state'
@ 6,20 say 'Function? Read (R) or Change (C)? ' get function
read
store !(function) to function
store function+'........' to keyboard

do while function='C'
  store ' ' to f1,f2,f3,f4,f5,f6,f7,f8
  @ 10,20 say 'Insert mode      on (1) or off (0)? ' get f1
  @ 11,20 say 'Caps   lock      on (1) or off (0)? ' get f2
  @ 12,20 say 'Num    lock      on (1) or off (0)? ' get f3
  @ 13,20 say 'Scroll lock      on (1) or off (0)? ' get f4
  @ 14,20 say 'Alt   shift pressed (1) or not (0)? ' get f5
  @ 15,20 say 'Ctrl  shift pressed (1) or not (0)? ' get f6
  @ 16,20 say 'Left  shift pressed (1) or not (0)? ' get f7
  @ 17,20 say 'Right shift pressed (1) or not (0)? ' get f8
  read
  store function+f1+f2+f3+f4+f5+f6+f7+f8 to keyboard
  store ' ' to function
enddo
*
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
* Set call address to subloc
set call to subloc
*
* Call the subroutine
call keyboard
*
* Display returned value
do while function='R'
  @ 8,20 say 'The keyboard state is'
  @ 8,42 say keyboard
  @ 10,20 say 'Insert mode'

  if $(keyboard,2,1)='1'
    @ 10,35 say 'on'
  else
    @ 10,35 say 'off'
  endif

  @ 11,20 say 'Caps Lock'
  if $(keyboard,3,1)='1'
    @ 11,35 say 'on'
  else
    @ 11,35 say 'off'
  endif

  @ 12,20 say 'Num Lock'
  if $(keyboard,4,1)='1'
    @ 12,35 say 'on'
  else
    @ 12,35 say 'off'
  endif

  @ 13,20 say 'Scroll Lock'
  if $(keyboard,5,1)='1'
    @ 13,35 say 'on'
  else
    @ 13,35 say 'off'
  endif

  @ 14,20 say 'Alt Shift'
  if $(keyboard,6,1)='1'
    @ 14,35 say 'pressed'
  else
    @ 14,35 say 'not pressed'
  endif

  @ 15,20 say 'Ctrl Shift'
  if $(keyboard,7,1)='1'
    @ 15,35 say 'pressed'
  else
    @ 15,35 say 'not pressed'
  endif

  @ 16,20 say 'Left Shift'
  if $(keyboard,8,1)='1'
    @ 16,35 say 'pressed'
  else
    @ 16,35 say 'not pressed'
  endif

  @ 17,20 say 'Right Shift'
  if $(keyboard,9,1)='1'
    @ 17,35 say 'pressed'
  else
    @ 17,35 say 'not pressed'
  endif

  store ' ' to function
enddo

wait to return
return
```


Brand New From Peter Norton

A PROGRAMMER'S EDITOR

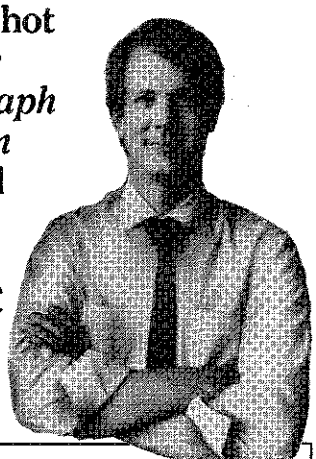
that's *lightning fast* with the *hot* features programmers need

A WONDERFUL EDITOR FOR PROGRAMMERS FROM PETER NORTON

THE NORTON EDITOR

only **$50**

Easily customized and saved ▲ *Split-screen editing* ▲ *A wonderful condensed/outline display* ▲ *Structured programming features:* ● *Auto-indenting for* Pascal *and* C ● *A hot "find matching punctuation" feature* ▲ *Great word features too:* ● *Paragraph reformat* ● *Word wrap, word action* ● *You can write your programs* and *your manuals with this one editor* ▲ *Mouse support, too* ▲ *Great for* assembler, *fabulous for* Pascal *and* C



A STEAL AT $50!

Direct from the man who gave you *The Norton Utilities, Inside the IBM PC,* and the *Peter Norton Programmer's Guide.*

"This is the programmer's editor that I wished I'd had when I wrote my *Norton Utilities.* Now I've got it for you to use. You can *program your way to glory* with *The Norton Editor.*"

Peter Norton

Peter Norton, 2210 Wilshire Blvd., #186
Santa Monica, CA 90403, 213-826-8032
Visa, MasterCard and phone orders welcome

Circle **17** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TWO MICROCOMPUTER GRAPHICS BOOKS

Review by J. Edward Volkstorf, Jr.

MicroComputer Displays, Graphics and Animation
by Bruce Artwick

As I read Bruce Artwick's *MicroComputer Displays, Graphics and Animation*, I felt as if I were reading the work of an old warrior, a veteran of a lot of battles. Actually I was: Bruce Artwick is the president of Sublogic Corporation, responsible for Flight Simulator, Night Mission Pinball, and their newest, Jet — all popular and very sophisticated microcomputer graphics games.

Little is left out of *MicroComputer Displays, Graphics and Animation*, and if there is a fault with the book, it is probably a tendency to overdo it. Nevertheless, this reviewer came away with a new appreciation for the subject of microcomputer graphics.

The nine chapters and three appendices add up to 374 pages. Hardware is covered first: displays and monitors, display controllers, and input/output devices (light pens, the mouse, graphics tablets, plotters, touch screens, and so on). Next is an introduction to interactive and simulation graphics, followed by a lengthy chapter on the mathematics of advanced graphics (coordinate systems, transformations, curve smoothing, and so on). A chapter describes animation and the problems with high speed animated displays. The book closes with a short chapter on business graphics, probably the weakest part of the book — a few thoughts on the power of business graphics and a simple algorithm to display bar graphs. Compared to the earlier chapters, this chapter seems like an afterthought.

The three appendices contain a table comparing foreign and domestic television frequency standards and a description of Apple II and IBM PC graphics — the latter discussion is basically a re-hash of the IBM Technical Reference manual.

The discussion of RGB versus composite color monitors is one example of the book's thorough treatment of its topics. A color chrome chart describes how colors are formed by mixing the basic three. This evolves into a rudimentary electronics circuit diagram for an RGB interface. Composite displays are then introduced, along with the problems of cramming a complex color image into a 5 MHz signal, which in turn leads to a treatment of "back porching" and "aliases." There is even a circuit diagram for converting RGB signals for a composite monitor.

The mathematics can become difficult. The theory of splines and manifolds is discussed. Various functions are introduced that describe how to convert a set of points into a smooth curve. The discussion can quickly get over one's head.

Artwick's discussion of milking every possible micro-second out of a processor is quite interesting — it includes a clever series of push the stack instructions to quickly clear the screen and a discussion of animation in a pin ball game.

Aside from occasional overkill, my only complaint is that attempts to produce graphics language standards — for example, the Graphics Kernel System (GKS)— are not mentioned at all.

If you want a broad picture of microcomputer graphics, you'd be well-advised to add this book to your library. The occasional heavy sledding is really not an obstacle and actually gives you a good feel for the scope of the field.

MicroComputer Displays, Graphics and Animation
by Bruce Artwick
Published by Prentice-Hall,
Englewood Cliffs, NJ, 1984
Price: $19.95

GKS Primer by Lucia McKay

Standards are a fact of life. In the computing field programming languages, data communication formats, and peripheral interface protocols are all areas where standards have been established. GKS stands for Graphics Kernel System and is an attempt to develop a graphics programming standard. Considering the explosive growth in the use of graphics in the last few years, this is not surprising — as soon as any field gets to a certain "critical mass," a group emerges and tries to standardize it.

GKS is essentially a two-dimensional, line-drawing interface recently adopted by the International Standards Organization (ISO) and soon to be adopted by the American Standards Institute (ANSI). Two other graphics standards also exist: Core, which was developed by the ACM's Special Interest Group in Graphics, and PMIG (Programmer Minimal Interface to Graphics). Each standard has its backers and detractors, but GKS appears to be the front runner with the widest industry support.

Think for a moment about the abundance of variables that come into play when you're writing graphics — the techniques for user interaction, the varieties of display hardware and processors, the languages that can be used. Defining a standard

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C PROGRAMMERS, CALL A POWER PLAY WITH . . .

. . . db__VISTA DATABASE MANAGEMENT SYSTEM FOR C.

db__VISTA is a full-featured programmer's DBMS. It handles your data powerfully, yet economically without the frills that make end-user DBMS's bulky, slow and expensive to license. Use only the features you need for maximum efficiency with minimum code and effort.

Powerful lineup of features. B-tree indexing, multiple key records, transaction processing, interactive database access utility, and file transfer utilities for dBASE, R:base and ASCII files. You even get 90 days applications development support free of charge.

Define your playbook up front. As a network model DBMS, **db__VISTA** is suited to applications development. A premium is placed on efficient use of disk storage, reduced data redundancy and fast access times allowing you to cross the goal line first.

It's your game plan. The database structure is specified by you in **db__VISTA's** data definition language (DDL). The DDL processor compiles the specification into tables (data dictionary) used by **db__VISTA's** library functions, which are called by your C program to manipulate and access the database.

db__VISTA's written in C so you can understand the signals. Source code is optional. No sweat. No royalties, either.

All this delivered for less than the price of season tickets.

Single user without source . . $195
Single user with source $495
Multi-user without source . . $495
Multi-user with source $990
Available for most popular C compilers under **MS-DOS, XENIX,** plus most **UNIX** systems.

Go for the power play and order **db__VISTA** now. Call (206) 747-5570 or

1-800-843-3313

at the tone touch 700-992. 30 day money-back guarantee.

RAIMA CORPORATION

12201 S.E. Tenth Street
Bellevue, WA 98005 USA
Telex: 9103330300

Circle **19** on the reader service card

C

HOW EASY DEVELOPING PROGRAMS CAN BE WITH THE C UTILITY LIBRARY!

COMPLETE. Over 300 tested and well documented functions. All features of the PC &AT are at your finger tips.

ADVANCED. Features like windows, data entry fields, switching displays, and batch file execution are supported.

GRAPHICS. A complete set of fast business graphics functions. Our low level graphics routines are the fastest in the business.

SOURCE. All source code is included. 95% of the library is written in C. Only functions demanding fast execution speed are in Assembler.

COMPATIBLE. With Lattice, Microsoft(3.0), CI-C86, Aztec, DeSmet, Wizard, and Mark Williams. All memory models are supported.

MUCH MORE. Dozens of string functions, the best time & date math and formatting, DOS directory and file mgmt., keyboard control, polled async communications, and more. **NO ROYALTIES.** We **SUPPORT** what we sell.

C UTILITY LIBRARY - $185
COMPILERS: Lattice $349, CI-C86 $329, Mark Williams $449. **Save $40-$50** with compiler & library package.

Specify compiler and version number when ordering. Add $4 for UPS or $7 for UPS 2-day shipping. NJ residents add 6% sales tax. VISA, MC, Chks & qualified PO's.

ESI Essential Software, Inc.
P.O. Box 1003
Maplewood, New Jersey 07040
914/762-6605

Circle **10** on the reader service card

methodology for graphics programming is an awesome task. GKS, an attempt to define that standard, is made up of an equally awesome 200 different functions and subroutines.

Because the standard is a relatively recent development, there are not that many books available on the subject. *GKS Primer* was published in 1984 by NOVA International and provides a good overview and introduction to GKS. NOVA International is not a publishing house, but a vendor of a software package called NOVA*GKS, which, you guessed it, is an implementation of GKS for various languages and mini/microcomputers.

NOVA International's book might be expected to contain a rather large sprinkling of NOVA*GKS related material, but this is not the case. There are hardly any references to the package. Instead, the author has done a commendable job of showing the key GKS concepts in a quick and lucid manner. The book is a little less than 100 pages, but highlights the main areas of GKS quite well. A list of references provides the reader with

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

YOU NEED A GOOD LIBRARY

POWER PACKS™

COMPLETE SOURCES
NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
— PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." — Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I (EXE files)** $99
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC. SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

Circle 23 on the reader service card

sources for more details.

The author, Lucia McKay, worked with Clinton N. Waggoner of NOVA International in writing the book. Ms. McKay is the owner of Human HiTech, which produces documentation and training material for software.

The book opens with an overview of GKS and a discussion of the need for a standard. The GKS primitives "polyline," "polymarker," "fill area," and "text" are then described. Polyline draws a set of connected line sequences based on the addresses in two arrays passed to it — like PC BASIC's LINE statement when used in a FOR/NEXT loop. Polymarker draws a set of points marked by some designated symbol such as a crosshair or dot. Fill area essentially defines an area, or polygon such as polyline, but with its interior painted or filled in some specified manner. The text primitive displays character symbols in various forms, positions, and directions.

A short FORTRAN program is developed using these primitives to create the line image of an aardvark. In later chapters this image is used to illustrate the more advanced features in GKS, such as coordinate transformation, rotation, clipping, panning, and zooming — a whimsical tour of GKS.

The book closes with a chapter on workstations. In GKS a workstation is defined as a display area with zero, one, or more input devices. For example, a color monitor with keyboard has a single display and input device. A plotter is a display with zero output devices. A number of functions that illustrate how hardware dependent characteristics can be handled under GKS are briefly described.

The only fault I find with the *GKS Primer*, and admittedly this is a minor complaint, is the lack of a complete GKS function and subroutine list. A final chapter in the book lists only those supplied in NOVA* GKS (about 193 in all). An even more minor point is the lack of parameter descriptions for the subroutines and functions in the list. This would have been a good reference source for the more technically oriented reader. Of course, if you are that interested, you can purchase the proposed standard directly from the International Standards Organization.

On the other hand, the book does provide a comprehensive graphics glossary. It includes both routine graphics terms as well as those relating strictly to GKS.

Anyone working in the development of computer graphics should find the book of interest. If you are just getting involved in the problems of graphics portability, it is highly recommended. GKS may be your answer to avoiding graphics software that is tightly tied to a specific set of graphics hardware.

GKS Primer
by Lucia McKay
Published by NOVA Graphics International, Austin, TX, 1984
Price: $ 9.95

Official Portrait of Ferengetti VI, believed to be the first Stand alone.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Massively Parallel Processor
edited by J.L.Potter
304 pages
The MIT Press, 1985

Nature seems to have invested billions of person-years in what appears to be a massively parallel architecture, and it just might lie in the future for computer programmers as well, and not just between the ears. This is a series of articles on the software and hardware of the MPP, the Massively Parallel Processor, a 16,384 processing-unit machine designed by Goodyear Aerospace for NASA, primarily for two-dimensional processing of satellite imagery. The MPP is handled on the front end by a PDP-11/34A or a VAX 11/780. It processes two-dimensional data with an "array unit" consisting of a number of 16,384 element planes, each a square of 128 rows and 128 columns. Each element in the plane can be, arbitrarily, a bit, or a floating point number. Instructions operate on the whole plane of data in parallel. The MPP can manage 6,553 million operations per second on 8-bit numbers.

The machine supports Parallel Pascal, which attempts, (what else?) to be machine independent. Given

```
var a,b,c: array[1..10] of integer;
```

then

```
for i:=1 to 10 do a[i]:=b[i]+c[i]+1;
```

in Parallel Pascal is

```
a:=b+c+1;
```

Given

```
var a,b: array[1..5,1..10] of integer;
```

the Parallel Pascal statement

```
a[,1]:=b[,4];
```

assigns the fourth column of b to the first column of a. More examples of Parallel Pascal:

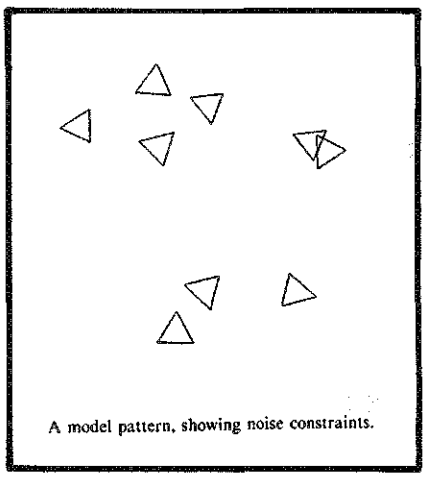
```
a:=shift(b,0,3);
c:=rotate(d,3);
```

Data Structures with Abstract Data Types and Pascal
Daniel F. Stubbs
459 pages
Brooks/Cole Publishing, 1985

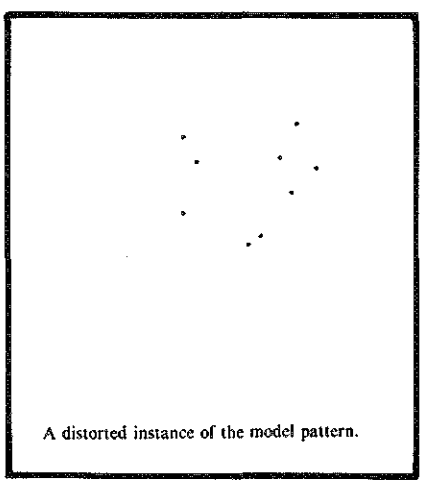
Readable implementations of linked lists, LIFO and FIFO stacks, Priority Queues, Disk Scheduler, Ring, AVL Tree, multiple stacks, etc. Specified in structured English, and then Pascal.

Model-Based Image Matching Using Location
Henry S. Baird
103 pages
MIT Press, 1985

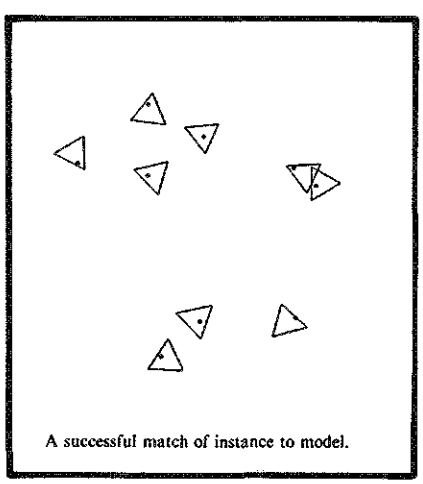
A doctoral thesis from 1984 that presents "an elegant solution to the problem of matching a known 'model' shape with an observed image of it. The assumption is that by translating, rotating and scaling, one can match features found in the image with features of the model."



A model pattern, showing noise constraints.



A distorted instance of the model pattern.



A successful match of instance to model.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

And now, from slightly out in left field:

Gate Arrays
John W Read, Editor
349 pages
McGraw Hill, 1985

A series of articles covering the basics of the technology. A Gate Array is a "semi-custom" chip that can typically replace twenty to fifty "glue chips," the small integrated circuits that are necessary to "glue" together, or interrelate large chips (such as MPUs and disk controllers).

Gate arrays owe their popularity to the economic advantage of their manufacture: The basic, underlying layers of a gate array are a generic pattern of logic gates. For the customer, a customized metal pattern is then overlayed, which "programs" or "wires-up" gates to form a final chip. The result is a less efficient use of silicon real-estate than a totally custom chip, but it is cheaper. A lot of board space is also saved. The book attempts to do a thorough survey of a fast-changing field in order to give a feel for the trade-offs, economic and otherwise, involved in selecting a gate array.

Chapters include "Basic IC Technology," "Bipolar and High Speed Arrays," "MOS/CMOS Arrays," "Analog/Lineal Master Slices," "Computer Aided Design Tools," "Gate Array Manufacture," "Test Equipment," "Application Considerations," and "Future Trends."

Atron's PC/AT Bugbusters

Hardware-assisted Software Debuggers for Bullet-proof PC/AT-based Products

A BUGBUSTER STORY

Brad Crain, a project manager at Software Publishing (the people who developed both PFS:WRITE and PFS: FILE), relates the following: "On Friday, March 22, 1985, I was about to get on an airplane with Jeff Tucker, who was co-author of PFS:WRITE with me, and fly to IBM's Boca Raton, Florida facility. For a week, we had been unsuccessfully trying to isolate a bug in a new software product. In a last, desperation move, I set up an early-Saturday morning appointment with ATRON.

"Three of us walked through ATRON's door at 8:00 the next morning. Using ATRON's hardware-assisted debugging tools, we had the problem identified and fixed by 10:30AM."

Mr. Crain concludes: "We'd never have found the bug with mere software debuggers, which have the bad habit of getting over-written by the very bugs they're trying to find. It doesn't surprise me that almost all the top-selling software packages were written by ATRON customers. Now that they've broadened their PC family of debuggers to include a PC/AT debugging tool, those of us seriously into 80286 development are greatly relieved."

ARE YOU TRYING TO DO SOMETHING SCAREY?

Like developing your AT-based software product in the dark? Without professional debugging tools?

Seven of the ten top-selling software packages listed by the *THE WALL STREET JOURNAL** were produced by ATRON customers. The PC PROBE™ bugbuster ($1595) accounts for much of this success. Now that the PC/AT is the new standard for advanced commercial and scientific development, ATRON is proud to announce the AT PROBE™ bugbuster ($2495). It has even more debugging capabilities than the PC Probe.

HOW BUGBUSTERS KEEP YOU FROM GETTING SLIMED

The AT PROBE is a circuit board that plugs into your PC/AT. It has an umbilical which plugs into your 80287 socket and monitors all processor activity.

Since AT PROBE can trace program execution in real time, and display the last 2048 memory cycles, you can easily answer the questions: "How did I get here?" and "What *are* the interrupts doing?"

It can solve spooky debugging problems. Like finding where your program overwrites memory or I/O - impossible with software debuggers.

You can even do source-level debugging in your favorite language, like C, Pascal or assembler. And after your application is debugged, the AT PROBE's performance-measurement software can isolate your application's bottlenecks.

Finally, the AT PROBE has its own 1-MByte of memory. Hidden and write-protected. How else could you develop that really large program, where the symbol table would otherwise demand most of your PC/AT memory.

BORLAND'S PHILIPPE KAHN: "THERE WOULDN'T BE A SIDEKICK™ WITHOUT ATRON'S DEBUGGERS."

So why waste more time reading though your program listing for the ten thousandth time, trying to find why your program starts howling with every full moon. Be like BORLAND, get your Atron bugbuster today and bust bugs tomorrow.

ATRON
THE DEBUGGER COMPANY
20665 Fourth Street • Saratoga, CA 95070 408/741-5900

*WSJ, June 24, 1985, reporting Softsel figures. © 1985 by ATRON. PC PROBE™ and AT PROBE™ ATRON. SIDEKICK™ Borland. IBM Corp. owns numerous trademarks. Ad by TRBA.

Circle 1 on the reader service card

Methods
(A Smalltalk Implementation)
Digitalk Inc.
5200 W. Century Blvd., Suite 250
Los Angeles, CA 90045

Requires 512K and two drives. Source code included.

A subset of Smalltalk-80 designed to run on the P C Digitalk seems to have invested a lot in this package. The 552-page manual, which is clearly laid out and typeset, consists of four parts: The first describes how to use Methods; the second presents the syntax and semantics of the Smalltalk language kernel; the third describes the classes that are used most often in applications and presents two applications as illustrations of their use; and the fourth part is an encyclopedia of all the methods and classes supplied in Methods.

The manual says this about classes: "All knowledge in a program is organized into a collection of *objects* — bundles of data and the algorithms which operate on that data. Each object has a set of *methods* (algorithms) for performing computations, and uses *messages* to ask other objects to perform their methods. Objects are grouped into *classes* according to their characteristics. All objects of the same class share the same methods. Classes are arranged in a hierarchy. This hierarchy allows objects to inherit

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

methods from other classes further up in the hierarchy. If an object receives a message for which it has no appropriate method, it can pass the message up the hierarchy until a method is found for handling the message."

TLC-LISP
The Lisp Company
P.O. Box 487
Redwood Estates, CA 95044
(408) 354-3668

Requires 256K, 150K bytes of source code, an 8086 compiler, and a WordStar style editor included.

Available as a whole for $250, or in four various subsets.

Is this version of LISP a labor of love? Repeated calls to the company were met by an answering machine — no eager marketing department to answer the phone. A few days later, a sleepy and distracted-sounding John Allen, one of the company's founders, returned the calls, talking for a while about what LISP "apparently" is, and muttering something about being short of money. He said he hadn't read the recent glowing review (October 1985) of TLC-LISP in *Byte*.

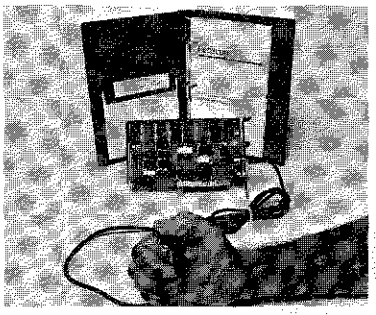
Allen says he was an orginal participant in Stanford's Artificial Intelligence Project and is the author of *Anatomy of LISP*. He has been involved in AI since 1963, when he ported LISP 1.5 from the IBM 704 to the IBM 7040.

TLC-LISP appears to be the most complete micro version available. It has a Smalltalk-like class sytem, as well.

The manual is excellent. It has this to say about First-Class Objects: "Even in the world of 'abstract data,' many modern languages make very concrete demands on the behavior of data objects. Those demands typically mean that each object be predefined . . . before a computation is begun. . . . Can you assign an array as value to a variable? Can you create an array as the value of a function? No. The only data objects that have this flexibility in these languages are numbers. Yet even here, constraints. . .are imposed because of hardware limitations [and]. . .difficulties in implementation. . . . What if we required that a natural language conversation begin with a definition of all words to be used in the conversation? . . .We are asking that the level of computational language be raised so that we can use [interactive, 'unstructured'] notation to describe our problems." □

Famous Computer Personalities #57
D. Leo Bishop, experiments with the original "line feed"

Periscope Delivers Professional Debugging Power



Models I and II are the same except Model I includes a write-protected RAM board. If you buy Model II, then find you need the board's protection, you can trade up to Model I. Both models include the break-out switch.

GET YOUR PROGRAMS WORKING FAST! "A problem we had for three weeks was solved in three hours," says Wade Clark of MPPi, Ltd. The break-out switch lets you stop your system anytime, even when it's hung, to see what's going on. Symbols, plus source debug for high-level languages, save you hours. Powerful breakpoints on registers, memory, I/O ports, and code help you find bugs fast.

MAKE YOUR SOFTWARE RELIABLE! "I can't live without it!! BRIEF, a text editor my company wrote, would not be as stable as it is today without Periscope," writes David Nanian. Periscope helps you find ALL your bugs so your software is reliable. The latest release includes traceback, which shows up to 2,016 previous instructions, EGA support, user exits, text file viewing, 8087/80287 status display, and much more.

DEBUG PROGRAMS OTHER DEBUGGERS CAN'T! Debug memory-resident programs, keyboard routines, interrupt-driven programs, device drivers, and DOS. Periscope gives you the best crash recovery capability available. "Periscope is rock solid," says user Doug Roberts.

GET THE BEST VALUE! Time and again users tell us Periscope pays for itself in a few hours after they begin using it. This means no professional software developer can afford NOT to try it!

PERISCOPE I: Board, Switch, Software, Manual, Reference Card ..... ONLY $295

PERISCOPE II: Switch, Software, Manual, Reference Card ..... JUST $145

PERISCOPE REQUIRES: An IBM PC, XT, AT or close compatible; DOS 2.0 or later; 128K RAM; one Disk Drive; and an 80-column Monitor.

Call Now, Toll-Free, For More Information Or To Place Your Order

MasterCard **800-722-7006** VISA

30-Day Money-Back Guarantee

Get Your Programs Up and Running;

UP PERISCOPE!

Data Base Decisions • 404 256-3860
14 Bonnie Lane • Atlanta, GA 30328

Circle **6** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

PJ Billboard *is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer. The goal is to provide our readers with a wide range of information on current products and updates.* ***PJ Billboard*** *serves our advertisers by providing a low cost, flexible format in which they can present their products to our readers.*

The standard layout is four columns, 9 picas wide, with copy set in 9 point on 10 point leading. ***PJ Billboard*** *offers a 400 character product description, including your company name, address, and telephone number. Boldface and italics available.*

The listings are $65, pre-paid, non-commissionable for a one time insertion. Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the ***Programmer's Journal*** *Advertising Department at 503-484-0602 for more information.*

FREE PC UNIX
With all purchases of $250 or more of software sold by ENTELEKON. Offer good until February 26, 1986. See our full page ad in this issue, page 13.
ENTELEKON
12118 Kimberley
Houston, TX 77024
(713) 468-4412 □

SEE™ A FULL SCREEN EDITOR $50. The same excellent editor we package with DeSmet C. Handles files larger than memory and can edit two files simultaneously. Its macro facility allows you to capture a series of keystrokes and replay them to ease repetitive tasks. Also featured: auto indent & brace/bracket/parenthesis matching to ease program entry. For the IBM. Note: For $59 more, you can have DeSemt C too! VISA, MC, AMEX.
C Ware Corporation
Dept. PJ
505 W. Olive, Suite "6"
Sunnyvale, CA 94086
(408) "20-9696 □

BASIC PROGRAMMERS Add **SALT & PEPPER** to your existing/new programs. Create dazzling Menus & Input screens like in dBASE & Lotus. Do Asc/desc. Sorts, PRT.SCR., Date processing, make your strings Roll and much more. **SALT & PEPPER** has 28 modules (in MSDOS compatible BASIC). They merge with your programs in minutes. **SALT & PEPPER** performs all the tricks, you get the applause. $59.95 ($2.50 s&h). Demo disk only $2.50 ppd MC/VISA.
COMPUTER GURU
40 Wagner Avenue
Piscataway NJ 08854
(201) 356-64"7 □

MAKE $50
Uses both built-in and user-supplied rules to specify how to construct your program(s). Compares directory dates and times to determine what files to update. Invokes compiler, assembler, linker or librarian to update files. Features: install your own built-in rules, extensive macro capabilities, use DOS 2.x/3.x path names. VISA, MC, AMEX.
C Ware Corporation
Dept. PJ
505 W. Olive, Suite "6"
Sunnyvale, CA 94086
(408) "20-9696 □

DESMET C $109
Full K&R C compiler for 8088/86/286 machines. Includes an assembler, linker, librarian, full screen deitor (SEE™), profiler, overlays and manual. Both 8087 and s/w floating point libraries included. PC/MSDOS-IBM format disks. *Rated 1st or 2nd* in all published benchmarks. Source level debugger $50 extra. DeSmet C for the Mac @ $150 includes machine-level debugger and SEE. VISA, MC, AMEX.
C Ware Corporation
Dept. PJ
505 W. Olive, Suite 767
Sunnyvale, CA 94086
(408) 720-9696

PERSONAL COMPUTER OWNERS CAN EARN
$1,000 to $5,000 monthly selling simple services performed by their computer. Work at home — in spare time. get free list of 100 best services to offer.
Write: C.I.L.D.D.
P.O. Box 60369
San Diego, CA 92106-8369.
□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

ERRATA

Arghh! We left a section of code from Neil Rubenking's article "Turbo Pascal 3.0 and Subdirectories" on the kitchen floor. Here's the missing part—put it at the very end.

```
var
  YourChoice : template_type;
begin
  if ParamCount > 0 then
    begin
      YourChoice := ParamStr(1);
      Get_every_file_that_matches(YourChoice);
    end
  else writeLn('Enter "WHERE <template>"');
end.
```

ARE YOU A FEW PAGES SHY OF A FULL ISSUE?

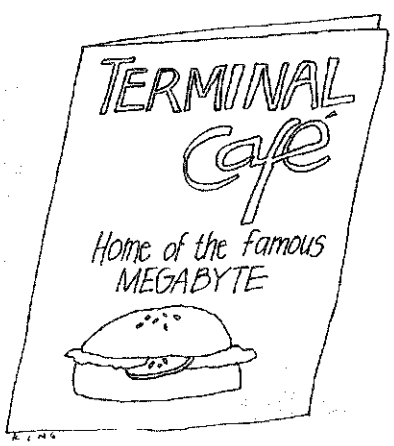
Several Copies of the last issue of *Programmer's Journal* were missing a signature. If your copy of *PJ* for November/December 1985 (Vol 3, Number 6) is a little light (see if pages 9-12 and 45-48 or 5-8 and 49-52 are missing), let us know, and we'll send you a replacement copy.

The Brett Salter routines in the last two issues had some difficulties. See this issue's section from Brett for details.

ADVERTISERS INDEX

CIRCLE NUMBER		PAGE NUMBER
1	Atron	50
2	Blaise	22
3	Borland	C4
4	Central Point Software	26
5	Dair Computer Systems	17
6	Data Base Decisions	51
7	Datalight	16
8	Ecosoft	30
9	Entelekon	13
10	Essential Software	47
11	Gimple Software	21
12	Landmark Software	24
13	Micromotion	44
14	Morgan Computing	41
15	Oliver Computing	25
16	Opt-Tech	9
17	Peter Norton Computing	45
18	Phoenix	4
19	Raima Corporation	47
20	Rational Systems	21
21	Softcraft	C2
22	Softfocus	40
23	Software Horizons	48
24	Software Studios	6 & 7
25	Turbopower Software	4

Patronize our advertisers. Along with you readers, they keep PJ *going. When you write or call to ask about their products or to order, let them know you saw their ad in* Programmer's Journal.



Main Menu from Boca Raton's favorite dive.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VINTAGE

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILEWEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBASE bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing relational database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and BASIC.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBASE II, Pascal's pointer power.
- Vol. 2, No. 6: Periscope—Building a hardware board for the IBM PC, AT&T surprises all, dBASE II to dBASE III?, Intel 80286 bugs by FNE, calendar functions for BASIC.
- Vol. 3, No. 1: November's Comdex, C—its uses and abuses, Using the C preprocessor, Memory segmentation examples, Index files (part 1), Structured programming language, Periscope, a symbolic debugger.
- Vol. 3, No. 2: UNIX! (again), Artificial intelligence, Doctor C's tips — pointers to functions, Calendar date routines in BASIC, Calculate record layouts, automatically!
- Vol. 3, No. 3: NAPLPS and the professional programmer, Instant-C, Printer control from a batch file, Repositioning the stars in Starfinder, Reducing EXE file size through data positioning, tools for better programming.
- Vol 3, No. 4: More UNIX stuff, C programming books, C code excerpts, How to cure the 'dreaded locked keyboard' condition, Calling MS-DOS from Turbo Pascal, DOS 2.X sort and find filters, the SET Tools.
- Vol. 3, No. 5: Windows done here; C programming books, Part Two; C code excerpts, Part Two; A menu-driven PC using DISPLAY.COM; Turbo Pascal and MASM: making them work together; dBASE II routines; Turbo Pascal 3.0; TopView Programmer's Toolkit.
- Vol. 3, No. 6: About C's HLL overhead, How to recover all your costs, Report on the COMDEX and Usenix shows, The Santa Cruz Operation XENIX for the IBM PC, Common C bugs CTOOLS2, Turbo Pascal 3.0 and subdirectories, More dBASE II routines, Clipper, A better way to interface BASIC and Assembler.

dj

PARALLEL PORTS

To order back issues, please use the form on the next page.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Yes! I want to start receiving programmer's journal

The Resource Journal for IBM PC Programmers

☐ Check enclosed Total amount _____

☐ Charge my Bankcard: ☐ VISA ☐ MC

_____ Expir. Date _____

_____ Signature

Please provide us with your name and address in the space below. It is not necessary to complete the Reader Service Form unless you wish to receive information on products from our Advertisers. We hope you will take the time to complete the Reader Survey on the back.

Back Issues are $5.50 ea. PREPAID ONLY

($6.50 Canada, $8.50 foreign, U.S. FUNDS ONLY)

Please send me the following Back Issues:

Vol. I # ___ ___ ___ ___ ___ ___

Vol. II # ___ ___ ___ ___ ___ ___

Vol. III # ___ ___ ___ ___ ___ ___

Bi-Monthly Subscription Rates

☐ New Subscription ☐ Renewal

Term	U.S.	Canada (U.S. funds only please)	Foreign
1 year 6 issues	☐ $24	☐ $30	☐ $40
2 years 12 issues	☐ $44	☐ $55	☐ $75
3 years 18 issues	☐ $64	☐ $80	☐ $110

Fold Here

NAME _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

TELEPHONE _____

For more information on the products advertised in *Programmer's Journal*, just fill out this postage paid self mailer and drop it in the mail. Our Advertisers will be glad to send you FREE information on the products you are interested in.

Filling out the Reader Survey on the back of this form will help us to serve you better.

Circle the number(s) below that correspond to the Reader Service number of the Advertiser from whom you would like information. Be sure to clearly print or type your name and address on the form at left.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31			
32	33	34	35	36	37	38	39	40	41	42	43	44	45			
46	47	48	49	50	51	52	53	54	55	56	57	58	59			
60	61	62	63	64	65	66	67	68	69	70	71	72	73			
74	75	76	77	78	79	80	81	82	83	84	85	86	87			
88	89	90	91	92	93	94	95	96	97	98	99	100				

GOOD UNTIL APRIL 30, 1986

Reader Information Card IV, 1

READER SERVICE CARD

Fold Here

_____

_____

_____

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 44 EUGENE, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

READER SURVEY

Program READER_SURVEY;
BEGIN
Check_appropriate_following_boxes;
Remove_page;
Fold_and_fasten;
Mail(Our_address_is_on_the_other_side);
END.

PJ needs your help to better understand this whole thing. If you fill out this survey, we'll really appreciate it.

Do you own a micro?

IBM Compatible ☐ Commodore ☐ Tandy ☐
Apple ☐ Atari ☐ TI ☐
HP ☐ Other _____

Do you use a microcomputer? How?

Educational ☐ Personal ☐ Professional ☐
Other _____

Do you use a main frame or mini?

Yes ☐ No ☐

How's your programming skill?

minimal ☐ medium ☐ too modest to say ☐

What languages do you use?

BASIC ☐ Assembler ☐ Pascal ☐
Modula-2 ☐ C ☐ Smalltalk ☐
Forth ☐ Prolog ☐ Lisp ☐
Logo ☐ Fortran ☐ dBASE ☐
Other _____

What languages would you like to use?

BASIC ☐ Assembler ☐ Pascal ☐
Modula-2 ☐ C ☐ Smalltalk ☐
Forth ☐ Prolog ☐ Lisp ☐
Logo ☐ Fortran ☐ dBASE ☐
Other _____

What's your age?

0-24 ☐ 25-40 ☐ 41-55 ☐
56-75 ☐ 76-90 ☐ 91- ☐

Do you . . .

Buy from newsstand ☐ Bookstore ☐
Read in the library ☐ Subscribe ☐
(don't let them see you rip out this card)
Keep back issues for reference ☐
Lend to friends ☐
Read the whole magazine ☐
Read parts only ☐

What would you like to see more of in PJ?

Reviews ☐ Code ☐ Languages ☐
Hardware ☐
Other _____

Are you interested in micros besides the PC?

Which ones? _____

What do you like best about PJ?

_____

_____

_____

What do you like least?

_____

_____

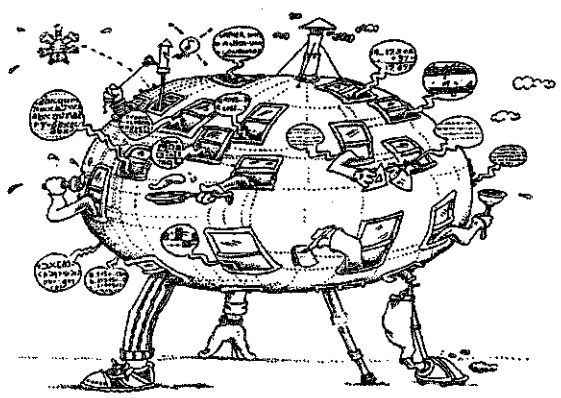
_____

What should we be asking you now?

_____

_____

_____



Our resourceful staff is diligently compiling this data.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SAVE OVER 30% ON OUR GIFT PACKS!
60-DAY MONEY-BACK GUARANTEE

How Borland's Three New Holiday Packs Will Fill Your Stocking Without Emptying Your Piggybank.

Three special packs with dazzling discounts that will help get you into a Holiday mood. You can get some of Turbo, most of Turbo, or all of Turbo—including the two newest members of the Turbo family, Turbo GameWorks™ and Turbo Editor Toolbox™. You also get our unmatched 60-day money-back guarantee, quality products that aren't copy-protected.

TURBO NEW PACK $95.00.

You get the two exciting new members of the Turbo Pascal family,

- TURBO GAMEWORKS, Chess, Bridge, and Go-Moku, complete with source code and a 200-page manual.
- TURBO EDITOR TOOLBOX, all the building blocks to make your own editors and word processors, complete with source code and a 200-page manual.

TURBO HOLIDAY PACK $125.00.

You get all three of the Turbo family classics for only $125.00 (about a 30% discount). Turbo Pascal 3.0 and Turbo Tutor and Turbo DataBase Toolbox—all for just $125.00.

- TURBO PASCAL combines the fastest Pascal compiler with an integrated development environment.
- TURBO TUTOR teaches you step-by-step how to use Turbo Pascal with commented source code for all program examples on diskette.
- TURBO DATABASE TOOLBOX offers three problem-solving modules for your Turbo Pascal programs: Turbo Access, Turbo Sort, and GINST, which generates a ready-to-run installation program that lets you forget about adapting your software to specific terminals.

TURBO HOLIDAY JUMBO PACK $245.00.

This is it—the whole thing, the entire Turbo family including its two newest members. You get:

- Turbo Pascal
- Turbo Tutor
- Turbo GameWorks NEW!
- Turbo Graphix Toolbox
- Turbo DataBase Toolbox
- Turbo Editor Toolbox NEW!

and you pay only $245.00 for all six! Which means that you're getting everything at only about $40 a piece. Quite a holiday deal. (And if you already own one or several members of the Turbo family, be creative—nothing can stop you from buying the Jumbo Pack, picking out the ones you already have and giving the rest as holiday gifts to family and friends. At these prices you can afford to give to others and to yourself.) Speaking of Holidays, this offer lasts until March 31, 1986. (At Borland, we like to make the Holidays last.)

NEW! **TURBO GAMEWORKS $69.95.**

Our new Turbo GameWorks offers games you can play and replay without Turbo Pascal or revise and rewrite with Turbo Pascal 3.0. We give you the source code, the manual, the diskettes and the competitive edge. Chess, Bridge and Go-Moku. State-of-the-art games that let you be player, referee, and rules committee all at once because you have the Turbo Pascal source code. Learn exactly how the games are made—so you can go off and make your own. And Turbo GameWorks is the only quality game you can buy that is not copy-protected. Sold separately, only $69.95. (Just $47.50 if you buy the Turbo New Pack.)

NEW! **TURBO EDITOR TOOLBOX $69.95.**

Build your own word processor—for only $69.95! You get ready-to-compile source code, a full-featured WordStar™-like word processor, and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. With Turbo Editor Toolbox, you can have the best of all word processors. You can make WordStar behave like Multi-Mate. Support windows just like Microsoft's Word. And do it as fast as WordPerfect does it. Incorporate your new "hybrids" into your programs to achieve incredible control and power. Sold separately, only $69.95. (If you buy the Turbo New Pack, the price drops to just $47.50.

NOT COPY-PROTECTED

Holiday Gift Packs, Turbo GameWorks™ & Turbo Editor Toolbox™

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133.

		Quantity
*Turbo Holiday Jumbo Pack	$245.00	
*Turbo Holiday Pack	$125.00	
*Turbo New Pack	$95.00	
Pascal	$69.95	
Pascal w/8087	$109.90	
Pascal w/BCD	$109.90	
Pascal w/8087 and BCD	$124.95	
Turbo DataBase	$54.95	
Turbo Graphix	$54.95	
Turbo Tutor	$34.95	
Turbo Editor	$69.95	
Turbo GameWorks	$69.95	

These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

*Gift Pack Offers Last Until March 31, 1986

Make checks payable to: Borland International.

The holiday packs include an upgrade coupon for both options so you get BCD and 8087 support for $39.95 (regularly $55.00).

Carefully describe your computer system!

Mine is: — 8-bit — 16-bit

I use: — PC-DOS — MS-DOS — CP/M-80 — CP/M-86

My computer's name and model is:

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Name:

Shipping Address:

City:

State: Zip:

Telephone:

NOTE: Turbo Editor Toolbox and Turbo GameWorks are available for the IBM PC and true-compatibles using Turbo Pascal 3.0 ONLY.

H36

Amount: (CA add 6% tax)

Payment: VISA MC Check Bank Draft

Credit Card Expiration Date: /

Card #

COD's and Purchase Orders will not be accepted by Borland International. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank.

BORLAND INTERNATIONAL

4585 SCOTTS VALLEY DRIVE, SCOTTS VALLEY, CA 95066 PHONE (408) 438-8400 TELEX 172373

Copyright 1985 Borland International BI-1017B

Turbo Pascal and Turbo Tutor are registered trademarks and Turbo DataBase Toolbox, Turbo Graphix Toolbox, Turbo Editor Toolbox, Turbo GameWorks, and MicroStar are trademarks of Borland International, Inc. WordStar is a trademark of MicroPro International Corp. Multi-Mate is a trademark of Multimate International Corp. Microsoft is a registered trademark and Word is a trademark of Microsoft Corp. WordPerfect is a trademark of Satellite Software International.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED